

# *RTI Connex<sup>t</sup>*

**Core Libraries and Utilities**

**Getting Started Guide**

**Addendum for Embedded Systems**

Version 5.0



Your systems. Working as one.



© 2012 Real-Time Innovations, Inc.  
All rights reserved.  
Printed in U.S.A. First printing.  
August 2012.

#### Trademarks

Real-Time Innovations, RTI, DataBus, and Connex are trademarks or registered trademarks of Real-Time Innovations, Inc. All other trademarks used in this document are the property of their respective owners.

#### Copy and Use Restrictions

No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form (including electronic, mechanical, photocopy, and facsimile) without the prior written permission of Real-Time Innovations, Inc. The software described in this document is furnished under and subject to the RTI software license agreement. The software may be used or copied only under the terms of the license agreement.

#### Technical Support

Real-Time Innovations, Inc.  
232 E. Java Drive  
Sunnyvale, CA 94089  
Phone: (408) 990-7444  
Email: support@rti.com  
Website: <https://support.rti.com/>

# Contents

## 1 Addendum for Embedded Platforms

## 2 Getting Started on Embedded UNIX-like Systems

2.1	Building and Running a Hello World Example.....	2-1
2.2	Configuring Automatic Discovery .....	2-2

## 3 Getting Started on INTEGRITY Systems

3.1	Building the Kernel .....	3-1
3.2	Building and Running a Hello World Example.....	3-2
3.2.1	Generate Example Code and Project File with rtiddsgen.....	3-2
3.2.2	Build the Publish and Subscribe Applications .....	3-3
3.2.3	Connect to the INTEGRITY Target from MULTI .....	3-3
3.2.4	Load the Application on the Target.....	3-3
3.2.5	Run the Application and View the Output.....	3-4

## 4 Getting Started on VxWorks 6.x Systems

4.1	Building the Kernel .....	4-1
4.2	Building and Running a Hello World Example.....	4-4
4.2.1	Generate Example Code and Makefile with rtiddsgen.....	4-4
4.2.2	Building and Running a Connex Application as a Kernel Task .....	4-5
4.2.3	Building and Running a Connex Application as a Real-Time Process.....	4-9

## 5 Getting Started on VxWorks 653 Platform 2.x Systems

5.1	Setting up Workbench for Building Connex Applications .....	5-2
5.2	Creating Connex Applications.....	5-2
5.3	Running Connex Applications on an Sbc8641d Target .....	5-13
5.4	Running Connex Applications on a SimPC Target.....	5-13

## 6 Getting Started on Wind River Linux Systems

## 7 Getting Started on Wind River VxWorks MILS 2.1.1 Systems

7.1	Step 1: Generate Connex Support Files and Example with rtiddsgen.....	7-2
7.2	Step 2: Create a VxWorks GuestOS Application Project.....	7-2
7.3	Step 3: Create a VxWorks MILS Integration Project.....	7-7
7.4	Step 4: Integrate GuestOS Application Project and Generated rtiddsgen Files into MILS Integration Project .....	7-12
7.5	Step 5: Deploy MILS Image to Target.....	7-14

# Chapter 1 Addendum for Embedded Platforms

In addition to enterprise-class platforms like Microsoft Windows and Linux, *RTI® Connex™* (formerly *RTI Data Distribution Service*) supports a wide range of embedded platforms. This document is a companion to the *RTI Core Libraries and Utilities Getting Started Guide* especially for users of those platforms. It describes how to configure some of the most popular embedded systems for use with *Connex* and to get up and running as quickly as possible. The code examples covered in this document can be generated for your platform(s) using the *rtiddsgen* code generator that accompanies *Connex*, as described in the *Getting Started Guide* ([Generating Code with rtiddsgen \(Section 4.3.2.1\)](#)).

This document assumes at least minimal knowledge with the platforms it describes and is not a substitute for the documentation from the vendors of those platforms. For further instruction on the *general* operation of your embedded system, please consult the product documentation for your board and operating system.

# Chapter 2 Getting Started on Embedded UNIX-like Systems

This chapter provides instructions on building and running *Connex* applications on embedded UNIX-like systems, including QNX® and LynxOS® systems. It will guide you through the process of generating, compiling, and running a Hello World application on an embedded UNIX-like system by expanding on [Generating Code with rtiddsgen](#) (Section 4.3.2.1) in the *Getting Started Guide*. Please read the following alongside that section.

In the following steps:

- ❑ All commands must be executed in a command shell that has all the required environment variables. For details, see [Set Up the Environment on Your Development Machine](#) (Section 3.1.1.1) in the *Getting Started Guide*.
- ❑ You need to know the name of your target architecture (look in your **NDDSHOME/lib** directory). Use it in place of `<architecture>` in the example commands. For example, your architecture might be 'i86Lynx4.0.0gcc3.2.2'.
- ❑ We assume that you have **gmake** installed. If you have **gmake**, you can use the generated makefile to compile. If you do not have **gmake**, use your normal compilation process. (Note: the generated makefile assumes the correct version of the compiler is already in your path and that **NDDSHOME** is set.)

---

## 2.1 Building and Running a Hello World Example

This section describes the basic steps for building and running an *rtiddsgen*-generated example on an embedded UNIX-like target.

1. Create a directory to work in. In this example, we use a directory called **myhello**.
2. In the **myhello** directory, create a file called **HelloWorld.idl** that contains a user-defined data type:

```
struct HelloWorld {
    string<128> msg;
};
```
3. Use the *rtiddsgen* utility to generate sample code and a makefile. Modify, build, and run the generated code as described in [Using Types Defined at Compile Time](#) (Section 4.3.2) in the *Getting Started Guide*:

For C++:

```
rtiddsgen -language C++ -example <architecture> HelloWorld.idl
gmake -f makefile_HelloWorld_<architecture>
./objs/<architecture>/HelloWorld_subscriber
./objs/<architecture>/HelloWorld_publisher
```

For Java:

```
rtiddsgen -language Java -example <architecture> HelloWorld.idl
gmake -f makefile_HelloWorld_<architecture>
gmake -f makefile_HelloWorld_<architecture> HelloWorldSubscriber
gmake -f makefile_HelloWorld_<architecture> HelloWorldPublisher
```

**For Java users:** The generated makefile deduces the path to the java executable based on the `APOGEE_HOME` environment variable<sup>1</sup>, which therefore must be set in order to run the example applications.

---

## 2.2 Configuring Automatic Discovery

In most cases, multiple applications—whether on the same host or different hosts—will discover each other and begin communicating automatically. However, in some cases you must configure the discovery service manually. For example, on LynxOS systems, multicast is not used for discovery by default; you will need to configure the addresses it will use. For more information about these situations, and how to configure discovery, see [Automatic Application Discovery \(Section 4.1\)](#) in the *Getting Started Guide*.

---

1. For example: `$(APOGEE_HOME)/lynx/pcc/ive/bin/j9`

# Chapter 3 Getting Started on INTEGRITY Systems

This chapter provides simple instructions on configuring a kernel and running *Connex* applications on an INTEGRITY system. Please refer to the documentation provided by Green Hills Systems for more information about this operating system.

This process has been tested on INTEGRITY 5.0.11 and assumes that applications are downloaded dynamically.

For more information on using *Connex* on an INTEGRITY system, please see the *RTI Core Libraries and Utilities Platform Notes*.

The first section describes [Building the Kernel \(Section 3.1\)](#).

The next section guides you through the steps to build and run an rtiddsgen-generated example application on an INTEGRITY target: [Building and Running a Hello World Example \(Section 3.2\)](#).

Before you start, make sure that you know how to:

1. Boot/reboot your INTEGRITY target.
2. Get the serial port output of your target (using telnet, minicom or hyperterminal).

---

## 3.1 Building the Kernel

Before you start, you should be familiar with running a kernel on your target.

1. Launch MULTI.
2. Select **File, Create new project**.
3. Choose the INTEGRITY Operating System and make sure the path to your INTEGRITY distribution is correct.
4. Choose a processor family and board name.
5. Click **Next**.
6. Choose Language: **C/C++**.
7. Project type: **INTEGRITY Kernel**.
8. Choose a project directory and name.
9. Click **Next**.
10. In Kernel Options, choose at least: 'TCP/IP stack'. Everything else can be left to default.

11. In the Project Builder, you should see the following file:

`<name of your project>_default.id` (under `src/resource.gpj`).

12. Right-click the file and edit it; the parameters of interest are the following:

```
CONSTANTS
{
    __INTEGRITY_DebugBufferSize = 0x10000
    __INTEGRITY_HeapSize = 0x100000
    __INTEGRITY_StackSize = 0x4000
    __INTEGRITY_DownloadSize = 0x400000
    __INTEGRITY_MaxCoreSize = 0x200000
}
```

Note that most *Connext* applications will require the `StackSize` and `HeapSize` parameters to be increased from their default value. The values shown above are adequate to run the examples presented in this document.

13. Once you have changed the desired values, right-click the top-level project and select **Build**.

14. Run the new kernel on your target.

## 3.2 Building and Running a Hello World Example

This section describes the basic steps for building and running an `rtiddsgen`-generated example on an INTEGRITY target:

- [Generate Example Code and Project File with `rtiddsgen` \(Section 3.2.1\)](#)
- [Build the Publish and Subscribe Applications \(Section 3.2.2\)](#)
- [Connect to the INTEGRITY Target from MULTI \(Section 3.2.3\)](#)
- [Load the Application on the Target \(Section 3.2.4\)](#)
- [Run the Application and View the Output \(Section 3.2.5\)](#)

### 3.2.1 Generate Example Code and Project File with `rtiddsgen`

To create the example applications:

1. Create a directory to work in. In this example, we use a directory called **myhello**.
2. In the **myhello** directory, create a file called **HelloWorld.idl** that contains a user-defined data type:

```
struct HelloWorld {
    string<128> msg;
};
```

3. Use the `rtiddsgen` utility to generate sample code and a project file as described in [Generating Code with `rtiddsgen` \(Section 4.3.2.1\)](#) in the *Getting Started Guide*. Choose either C or C++.

For C:

```
rtiddsgen -language C -example <architecture> HelloWorld.idl
```

For C++:

```
rtiddsgen -language ++C -example <architecture> HelloWorld.idl
```

In your **myhello** directory, you will see that *rtiddsgen* has created a number of source code files (described in Section 3.7 of the *RTI Core Libraries and Utilities User's Manual*), additional support files (not listed here), and a project file: **HelloWorld\_default.gpj**.

4. Edit the example code to modify the data as described in [Generating Code with rtiddsgen \(Section 4.3.2.1\)](#) in the *Getting Started Guide*.

### 3.2.2 Build the Publish and Subscribe Applications

1. In a plain text editor, edit the top-level project file that was generated by *rtiddsgen*, **HelloWorld\_default.gpj**, so that it points to the path to your INTEGRITY distribution:
  - For INTEGRITY 5 systems:  
Under **[Project]**, add the argument **-os\_dir=<path to your INTEGRITY distribution>**
  - For INTEGRITY 10 systems:  
Set **macro \_\_OS\_DIR=<path to your INTEGRITY distribution>**
2. Save your changes.
3. Launch MULTI.
4. Open the top-level project file, **HelloWorld\_default.gpj**, in MULTI:
  - For INTEGRITY 5 systems:  
Select **File, Open Project Builder**, then open the project file from there.
  - For INTEGRITY 10 systems:  
Select **Components, Open Project Manager**, then open the project file from there.
5. Right-click on the top-level project and build the project.

### 3.2.3 Connect to the INTEGRITY Target from MULTI

1. From the MULTI Launcher, click the Connection button and open the Connect option. Your mode should be Download (Download and debug application).
2. Create a custom connection with the following line:

For targets that only support the older INDRT connection mechanism:

```
rtserv -port udp@<ip address of your INTEGRITY target>
```

For targets that support the newer INDRT2 connection mechanism:

```
rtserv2 -port udp@<ip address of your INTEGRITY target>
```

(You might be able to see the IP address of your target on the output of its boot sequence.)

You only have to create your connection once, MULTI will remember it.

3. Make sure your target has booted; *then* select **Connect**. You should see a new window with the Kernel Tasks running on your target.

### 3.2.4 Load the Application on the Target

1. In the task window, select **Target, Load module**.
2. Browse for your executables; there should be 3 of them in your project directory:
  - **HelloWorld\_publisherdd**
  - **HelloWorld\_subscriberdd**
  - **posix\_shm\_manager**

3. Load the **posix\_shm\_manager** first, it will appear in the **Tasks** window as a separate address space and start running by itself once loaded. It will allow you to use the shared memory transport on your target.

Note: The default *rtiddsgen*-generated code tries to use shared memory, so unless you have manually disabled it, your application will crash if you do not load the shared memory manager before running the application.

4. Load the publisher, subscriber, or both. They should appear in separate address spaces in the Tasks window.

### 3.2.5 Run the Application and View the Output

1. Select the task called "Initial" in your application's address space in the Tasks window; you can either click the play button to run it, or click the debug button to debug it.

Note that with some versions of INTEGRITY, it is difficult to pass arguments to applications. Arguments can always be hard-coded in your application before compiling it. To quickly experiment with multiple runs of the application with different arguments, one option is to run your application within the debugger. Then you can set a breakpoint before the arguments are used and change them at that point.

2. From the Tasks window, select **Target, Show Target Windows**. This will show you the standard output of your target.

Some errors messages may still go through the serial port, so you should leave your serial port connection open and monitor it as well.

#### To reboot the target:

Go to your serial port connection monitor and type '**rset**'.

# Chapter 4 Getting Started on VxWorks 6.x Systems

This chapter provides simple instructions to configure a kernel and run *Connex*t applications on VxWorks 6.x systems. Please refer to the documentation provided by Wind River Systems for more information on this operating system. See also the *RTI Core Libraries and Utilities Platform Notes*.

This chapter will guide you through the process of generating, compiling, and running a Hello World application on VxWorks 6.x systems by expanding on [Generating Code with rtiddsgen](#) (Section 4.3.2.1) in the *Getting Started Guide*; please read the following alongside that section.

The first section describes how to build the kernel:

- ❑ [Building the Kernel \(Section 4.1\)](#)

The next section guides you through the steps to generate, modify, build, and run the provided example HelloWorld application on a VxWorks target:

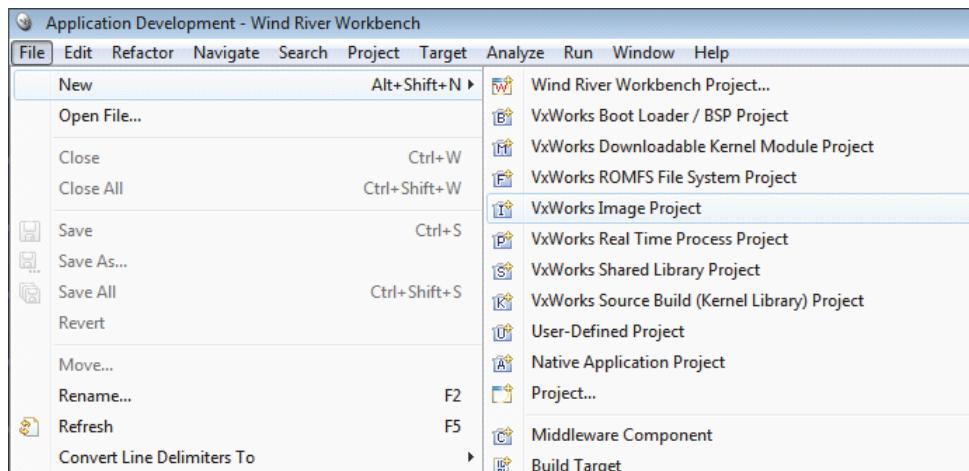
- ❑ [Building and Running a Hello World Example \(Section 4.2\)](#)

---

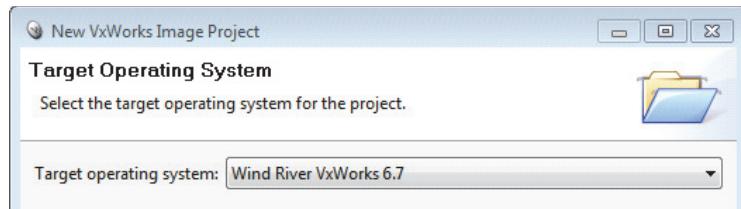
## 4.1 Building the Kernel

Before you start, you should be familiar with running a kernel on your target.

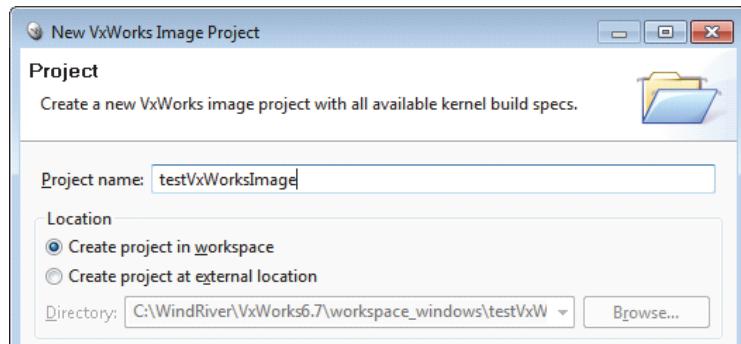
1. Launch Workbench.
2. Select **File**, **New**, **VxWorks Image Project**



3. Select the desired operating system; click **Next**.



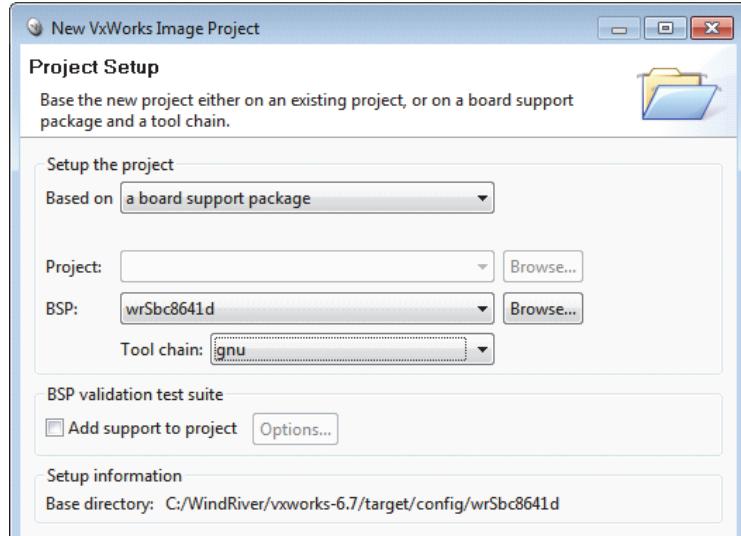
4. Give your project a name; click **Next**.



5. Choose the **board support package (BSP)** based on your hardware.

6. For VxWorks 6.9: Select the correct **Address mode**.

7. For the Tool chain option, select **GNU**; click **Next**



8. a) For VxWorks 6.8 and higher:

Select **SMP support in kernel** (regardless of whether or not you run on an SMP board).

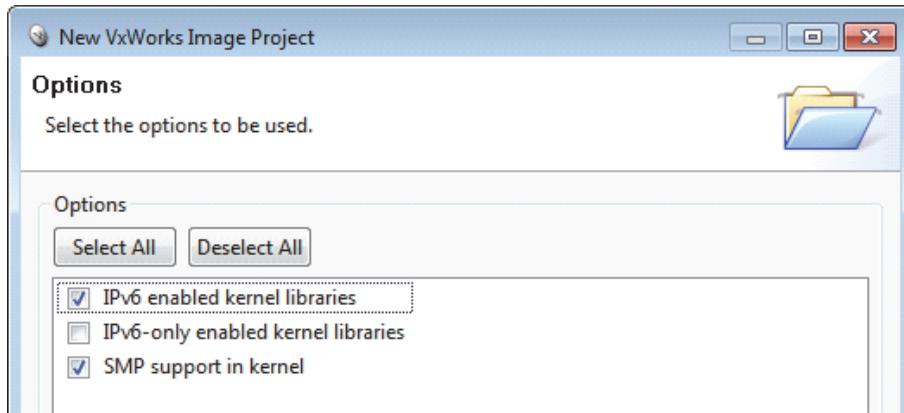
b) For VxWorks 6.7:

Select **SMP support in kernel** only if you want SMP support.

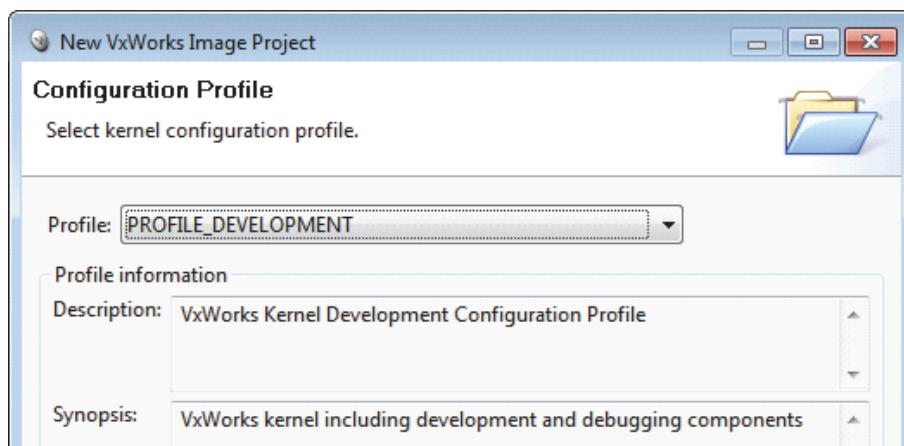
c) If your architecture supports IPv6:

Select **IPv6 enabled kernel libraries**; click **Next**.

To find out if your architecture supports IPv6, see the *Platform Notes (RTI\_CoreLibrariesAndUtilities\_PlatformNotes.pdf)*, Section 9.6 Supported Transports).



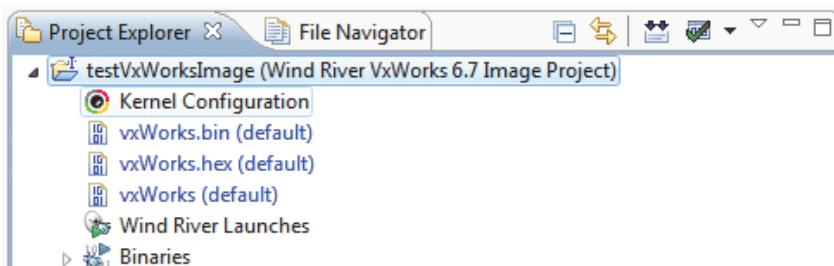
9. Select PROFILE\_DEVELOPMENT.



10. Leave everything else at its default setting. Click **Finish**.

Your project will be created at this time.

11. From the Project Explorer, open **Kernel Configuration**.



12. a) For VxWorks 6.8 and higher:

Add **Operating System Components**, **Kernel Components**, **\_thread variables support**.

b) For VxWorks 6.7, ONLY if you have enabled SMP support in the kernel:

Add **Operating System Components**, **Kernel Components**, **\_thread variables support**.

13. For VxWorks 6.4 and below, add the following modules:

- **ZBUF Socket** (under Network Components, Network Socket Components)  
The *Connext* libraries for VxWorks Kernel Mode use ZBUF sockets. If you do not add this module to the kernel, you will see undefined symbols when loading the *Connext* application on the target.
- **IGMP v4** (under Network Components, Network Protocol Components, Network IPv4 Components)  
This will enable multicast for the target.

14. If you plan on accessing your target via the network, you may need the following modules:

- **Telnet Server** (under Network Components, Applications, Telnet Components)  
This will allow you to telnet into the target.
- **NFS client all** (under Operating System Components, IO System Components, NFS components)  
This will allow you to see networked file systems from the target (contact your system administrator to find out if you have them set up).

15. If you are running applications in RTP mode, you may increase **Operating System components**, **Real Time Processes components**, **Number of entries in an RTP fd table** from the default value of 20 to a higher value such as 256. This will enable you to open more sockets from an RTP application.

16. Compile the Kernel by right-clicking the project and selecting **Build project**.

The Kernel and associated symbol file will be found in *<your project directory>/default/*.

---

## 4.2 Building and Running a Hello World Example

This section will guide you through the steps required to successfully run an *rtiddsgen*-generated example application on a VxWorks 6.x target using kernel mode or RTP mode:

- Kernel Mode:** see [Section 4.2.2](#)
- RTP Mode:** see [Section 4.2.3](#)

### 4.2.1 Generate Example Code and Makefile with **rtiddsgen**

To create the example applications:

1. Set up the environment on your development machine: set the NDDSHOME environment variable and update your PATH as described in [Step 1: Set Up the Environment \(Section 3.1.1\) in the Getting Started Guide](#).
2. Create a directory to work in. In this example, we use a directory called **myhello**.
3. In the **myhello** directory, create a file called **HelloWorld.idl** that contains a user-defined data type:

```
struct HelloWorld {  
    string<128> msg;  
};
```

4. Use the *rtiddsgen* utility to generate sample code and a makefile as described in [Generating Code with rtiddsgen \(Section 4.3.2.1\)](#) in the *Getting Started Guide*. Choose either C or C++.

**Note:** The architecture names for Kernel Mode and RTP Mode are different.

For C:

```
rtiddsgen -language C -example <architecture> HelloWorld.idl
```

For C++:

```
rtiddsgen -language C++ -example <architecture> HelloWorld.idl
```

Edit the generated example code as described in [Generating Code with rtiddsgen \(Section 4.3.2.1\)](#) in the *Getting Started Guide*.

## 4.2.2 Building and Running a Connexxt Application as a Kernel Task

There are two ways to build and run your *Connexxt* application:

- [Using the Command Line \(Section 4.2.2.1\)](#)
- [Using Workbench \(Section 4.2.2.2\)](#)

### 4.2.2.1 Using the Command Line

1. Set up your environment with the **wrenv.sh** script or **wrenv.bat** batch file in the VxWorks base directory.
2. Set the **DDDSHOME** environment variable as described in [Step 1: Set Up the Environment \(Section 3.1.1\)](#) in the *Getting Started Guide*.
3. Build the Publisher and Subscriber modules using the generated makefile. You may have to modify the **HOST\_TYPE**, compiler and linker paths to match your development setup.
4. To use dynamic linking, remove the *Connexxt* libraries from the link objects in the generated makefile.

(Note: steps 5-7 can be replaced by establishing a telnet connection to the VxWorks target. In that case, Workbench does not need to be used and both the Host Shell and Target Console will be redirected to the telnet connection. Once in the C interpreter (you will see the prompt '**>**' in the shell) you can type **cmd** and then **help** for more information on how to load and run applications on your target.)

5. Launch Workbench.
6. Make sure your target is running VxWorks and is added to the Remote Systems panel. (To add a new target, click the **New Connection** button on the Remote System panel, select **Wind River VxWorks 6.x Target Server Connection**, click **Next**, enter the Target name or address, and click **Finish**).
7. Connect to the target and open a host shell by right-clicking the connected target in the **Target Tools** sub-menu.
8. In the shell:

If you are using static linking: Load the **.so** file produced by the build:

```
>cd "directory"
>ld 1 < HelloWorld_subscriber.so
```

(Where 'directory' refers to the location of the generated object files.) If you are using dynamic linking: load the libraries first, in this order: **libnddscore.so**, **libnddsc.so**, **libnddscpp.so**; then load the **.so** file produced by the build.

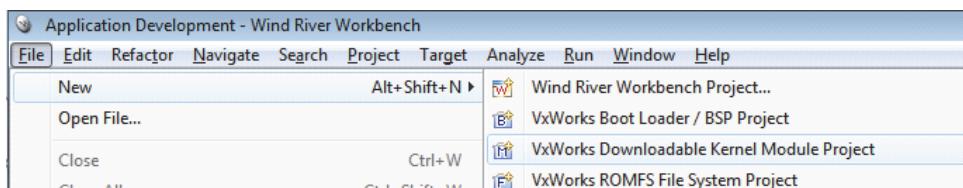
9. Run the **subscriber\_main** or **publisher\_main** function. For example:

```
>taskSpawn "sub", 255, 0x8, 150000, subscriber_main, 38, 10
```

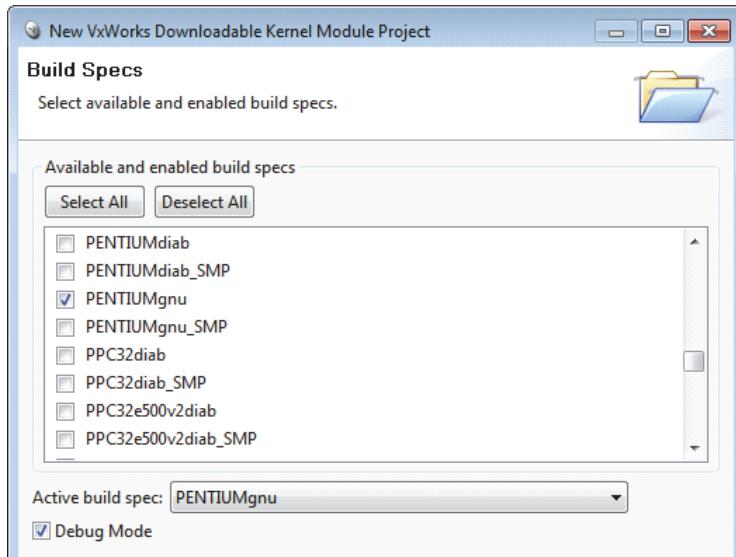
In this example, 38 is the domain ID and 10 is the number of samples.

#### 4.2.2.2 Using Workbench

1. Start Workbench.
2. Select **File**, **New**, **VxWorks Downloadable Kernel Module Project**.



3. Give your project a name; click **Next**.
4. Select the default options until you reach the dialog titled **Build Specs**. In this dialog, choose the desired build spec.

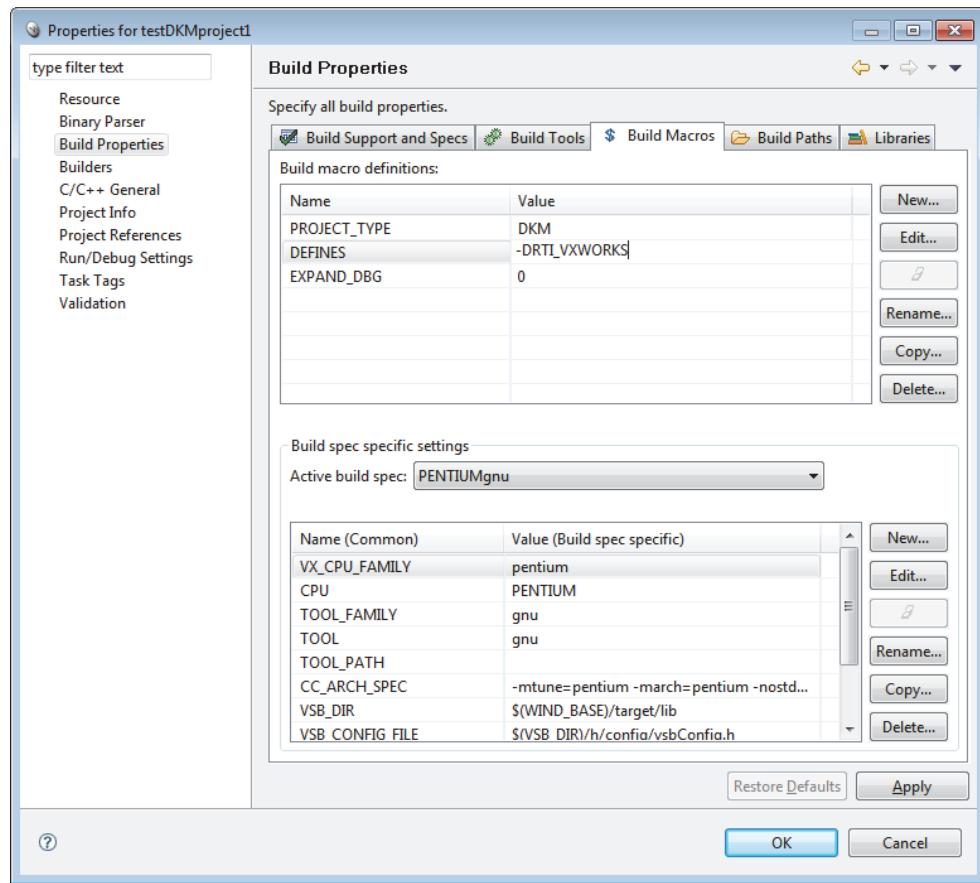


5. Leave everything else at its default setting; click **Finish**.

Your project will be created at this time.

6. Copy the source files and headers generated by *rtiddsgen* in [Section 4.2.1](#) into the project directory.
7. View the added files by right-clicking on the project in Project Explorer, then selecting **Refresh** to see the files.
8. Open the project Properties by right-clicking on the project in Project Explorer and selecting **Properties**.

9. In the dialog box that appears, select **Build Properties** in the navigation pane on the left.



10. a) In the Build Macros tab:

Add **-DRTI\_VXWORKS** to DEFINES in the Build macro definitions.

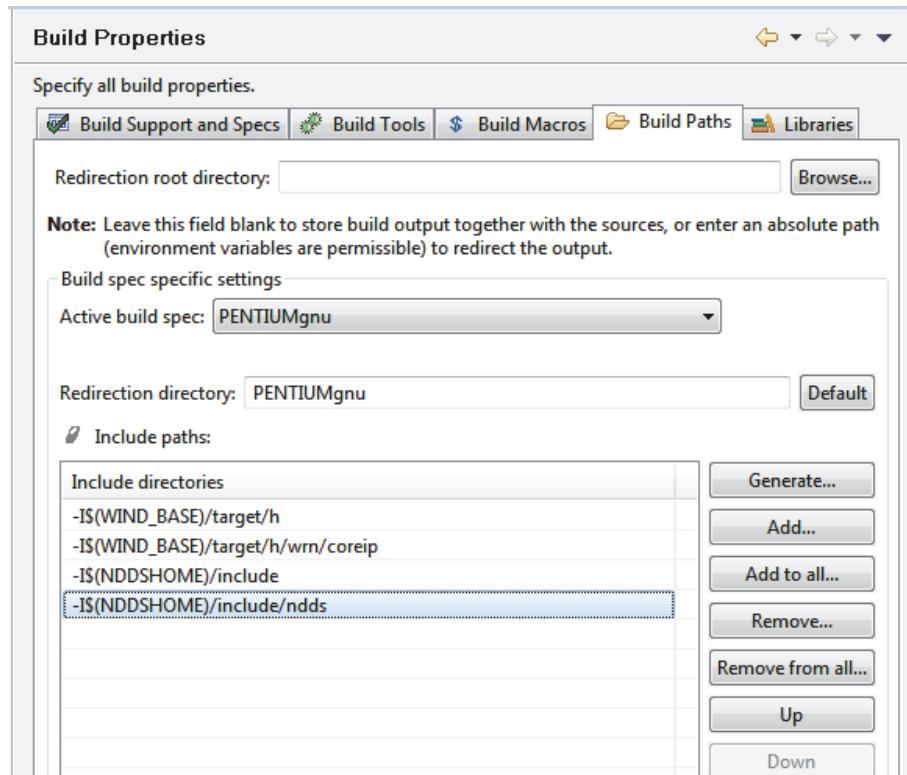
b) If you are using static linking, in the Variables tab:

- Add to LIBPATH: **-L\$(NDDSHOME)/lib/<architecture>**
- Add to LIBS: **-lnddscppz -lnddscz -lnddscorez** (in that order)

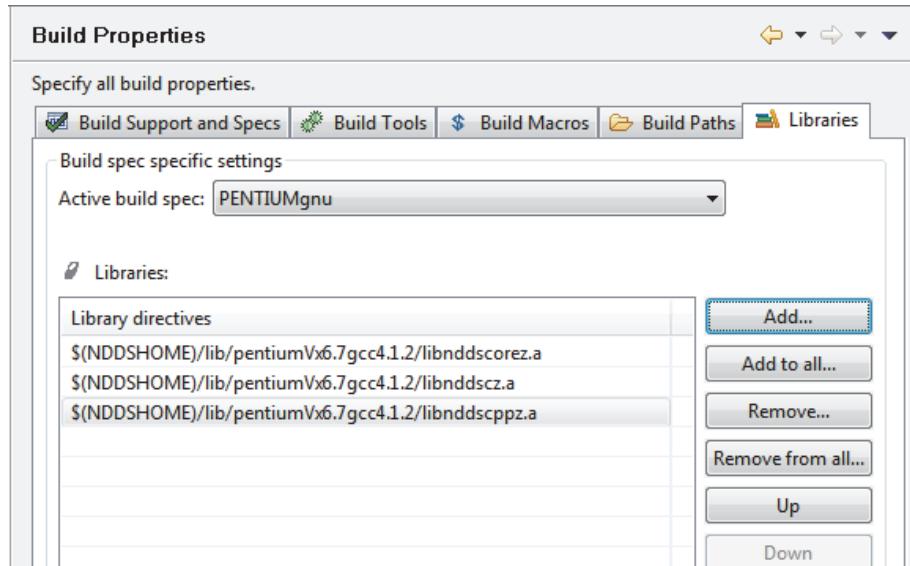
(If you are using dynamic linking, there are no changes required to LIBPATH or LIBS.)

11. In the Build Paths tab, add both of these:

- `-I$(NDDSHOME)/include`
- `-I$(NDDSHOME)/include/ndds`



12. If you are using dynamic linking: In the Libraries tab, add the Library directives shown below:



13. Click **Apply** to save the changes, then click **OK** to exit the Properties menu.
14. Build the project by right-clicking on the project in Project Explorer, then selecting **Build**.
15. Run the application as described starting in [Step 5 on page 4-5](#), except load **Hello-World.out** instead of **HelloWorld\_subscriber.so** when you get to [Step 9 on page 4-6](#).

## 4.2.3 Building and Running a Connext Application as a Real-Time Process

There are two ways to build and run your *Connext* RTP application:

- [Using the Command Line \(Section 4.2.3.1\)](#)
- [Using Workbench \(Section 4.2.3.2\)](#)

### 4.2.3.1 Using the Command Line

1. Generate the source files and the makefile with *rtiddsgen*.

**Note:** The architecture names for Kernel Mode and RTP Mode are different.

Please refer to the *RTI Core Libraries and Utilities User's Manual* for more information on how to use *rtiddsgen*.

2. Set up your environment with the **wrenv.sh** script or the **wrenv.bat** batch file in the VxWorks base directory.
3. Set the NDDSHOME environment variable as described in [Step 1: Set Up the Environment \(Section 3.1.1\) in the Getting Started Guide](#).
4. Build the Publisher and Subscriber modules using the generated makefile. You may need to modify the HOST\_TYPE, compiler and linker paths to match your development setup.

**Notes:**

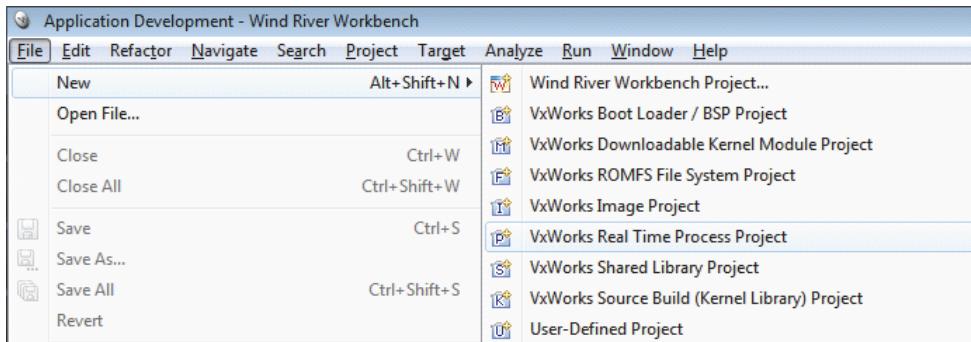
- Steps 5-12 below can be replaced by establishing a telnet connection to the VxWorks target. In that case, Workbench does not need to be used and both the Host Shell and Target Console will be redirected to the telnet connection. Once in the C interpreter (you will see a prompt '**->**' in the shell) you can type **cmd** and then **help** for more information on how to load and run applications on your target.)
- If you want to dynamically link your RTP to the RTI libraries (VxWorks 6.3 and above only), make the following modifications the generated makefile:

```
LIBS = -L$(NDDSHOME)/lib/<architecture> -non-static -lnddscpp \
       -lnddsc -lnddscore $(syslibs_<architecture>)
```

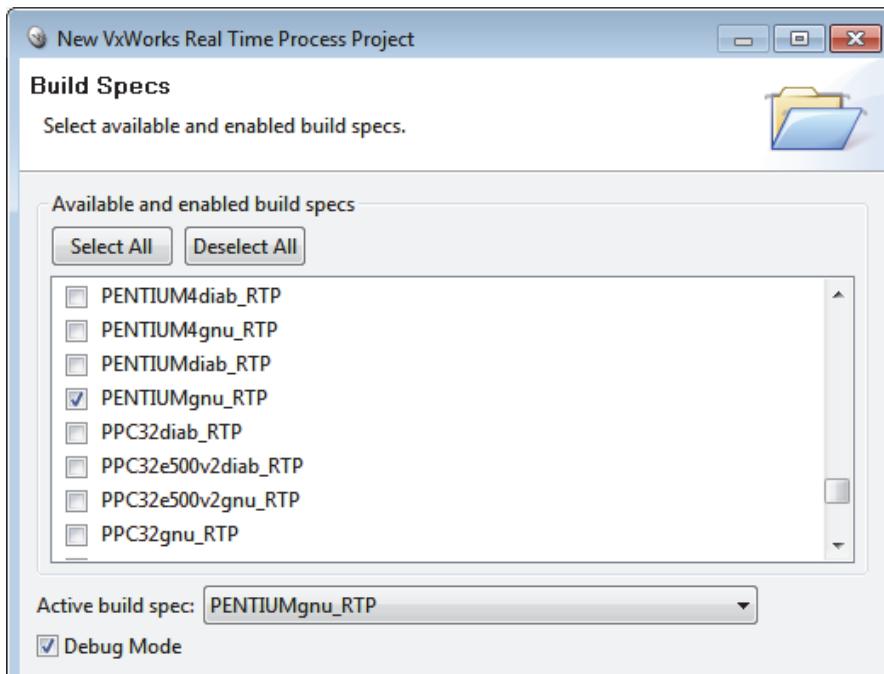
5. Add to the **LD\_LIBRARY\_PATH** environment variable the path to your RTI libraries as well as the path to **libc.so.1** of your VxWorks installation to launch your RTP successfully.
6. Launch Workbench.
7. Make sure your target is running VxWorks.
8. Connect to the target with the target manager and open a host shell and a Target Console Tool to look at the output. Both are found by right-clicking the connected target in the **Target Tools** sub-menu.
9. Right-click on your target in the Target Manager window, then select **Run, Run RTP on Target**.
10. Set the **Exec Path on Target** to the **HelloWorld\_subscriber.vxe** or the **HelloWorld\_publisher.vxe** file created by the build.
11. Set the arguments (domain ID and number of samples, using a space separator).  
A Stack size of 0x100000 should be sufficient. If your application doesn't run, try increasing this value.
12. Click **Run**.

#### 4.2.3.2 Using Workbench

1. Start Workbench.
2. Select **File**, **New**, **VxWorks Real Time Process Project**.

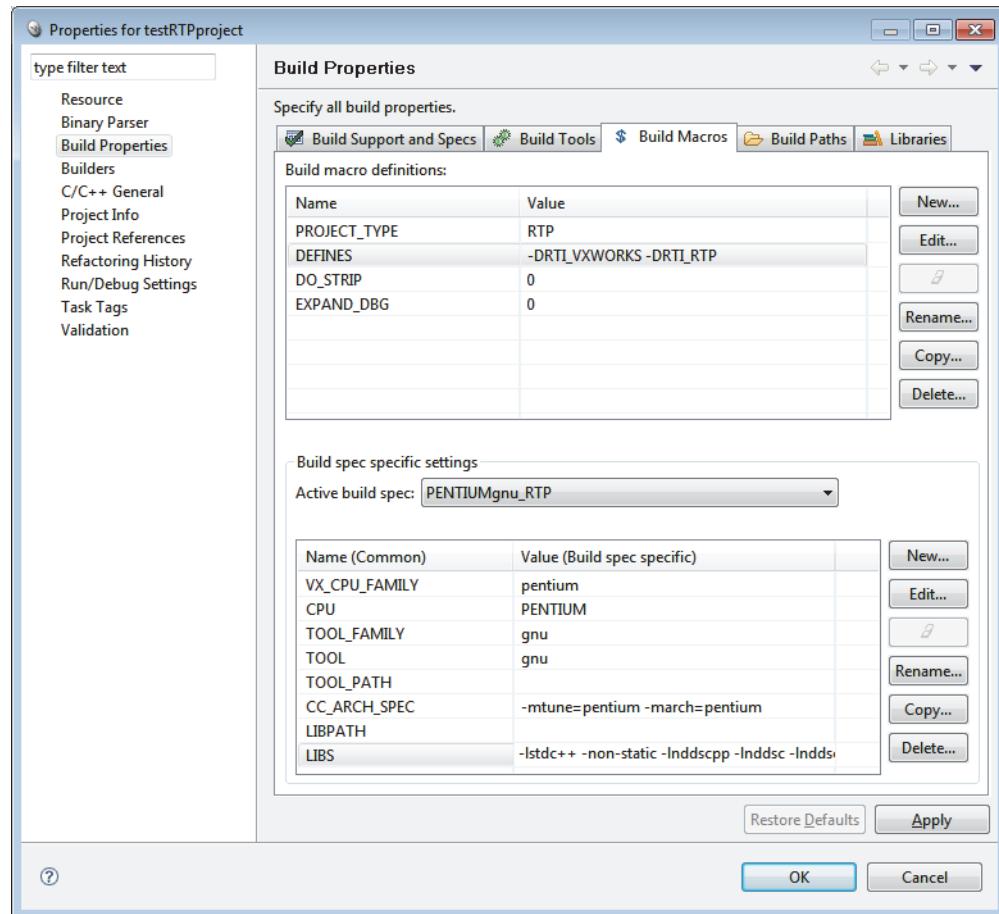


3. Give your project a name; click **Next**.
4. You can select the default options until you reach the dialog titled **Build Specs**. In this dialog, choose the desired build spec:



5. Leave everything else at its default setting; click **Finish**.  
Your project will be created at this time.
6. Copy the source and header files generated by *rtiddsgen* in [Section 4.2.1](#) into the project directory. There can only be one **main()** in your project, so you must choose *either* a subscriber or a publisher. If you want to run both, you will need to create two separate projects.
7. View the added files by right-clicking on the project in Project Explorer, then selecting **Refresh** to see the files.
8. Open the project Properties by right-clicking on the project in Project Explorer and selecting **Properties**.

9. In the dialog box that appears, select **Build Properties** in the navigation pane on the left.



10. a) In the Build Macros tab: Add **-DRTI\_VXWORKS -DRTI\_RTP** to DEFINES in the Build macro definitions.

b) If you are using *static* linking, in the Variables tab:

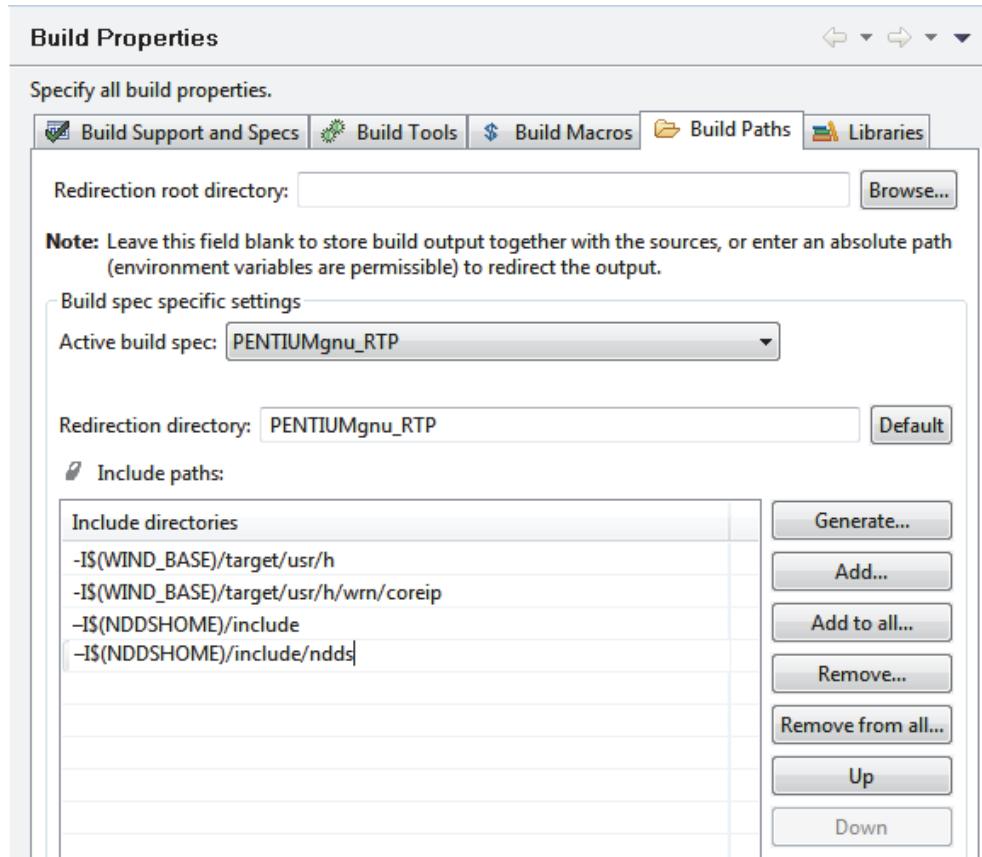
- Add to LIBPATH: **-L/(NDDSHOME)/lib/<architecture>**
- Add to LIBS: **-lnddscppz -lnddscz -lnddscorez** (*in that order*)

c) If you are using *dynamic* linking, in the Variables tab:

- Add to LIBS: **-non-static -lnddscpp -lnddsc -lnddscore** (*in that order*)

11. In the Build Paths tab, add:

- **-I\$(NDDSHOME)/include**
- **-I\$(NDDSHOME)/include/ndds**



12. Click **Apply** to save the changes, then click **OK** to exit the Properties menu.

13. Build the project, by right-clicking on the project in Project Explorer, then selecting **Build**.

14. Run the application as described starting in [Step 5 on page 4-9](#).

# Chapter 5 Getting Started on VxWorks 653 Platform 2.x Systems

This chapter provides simple instructions on how to configure a kernel and run *Connext* applications on a VxWorks 653 Platform 2.x system. Please refer to the documentation provided by Wind River Systems for more information, as well as the *RTI Core Libraries and Utilities Platform Notes*.

Developing a complete system typically involves the cooperation of developers who play the following principal roles:

- A platform provider*, who develops the platform
- An application developer*, who develops applications
- A system integrator*, who designs and specifies the module, and integrates a set of applications with a platform to create a module

For more information on these roles, please see the *VxWorks 653 Configuration and Build Guide*.

This document assumes the above distribution of development responsibilities, with the *Connext* Core Libraries being a part of the application. This document is targeted towards platform providers, application developers, and system integrators.

For **platform providers**, this chapter indicates what your system must provide to *Connext*. Platform providers must provide a platform that application developers will use to create the application. The provided platform must support worker tasks and the socket driver. For the actual list of components, refer to [Table 9.3, “Building Instructions for VxWorks 653 Architectures,” on page 47](#) in the *Platform Notes*.

For **application developers**, this chapter describes how to create *Connext* applications. Application developers must use the platform provided by the platform provider. To create a *Connext* application, follow [Step 6 on page 5-9](#) to [Step 8 on page 5-10](#).

For **system integrators**, this document describes how to combine the platform from the platform provider, and the application from the application developer, and create the system to be deployed. System integrators must create an integration project using the module OS and partition OS provided by the platform provider, and the application provided by the application provider. To create a system capable of running *Connext* applications, the system integrator needs to create a ConfigRecord considering the requirements noted in [Step 2 on page 5-6](#).

For **someone creating a Connext application**, this document provides an example from the ground up.

## 5.1 Setting up Workbench for Building Connex Applications

1. Install Workbench.
2. Install **partition\_socket\_driver\_v1.3**. Follow instructions from Wind River for the installation.

For this example, the following steps were used for the installation:

- a. Copy the socket driver files from Wind River to each BSP of interest. For example, for sbc8641Vx653-2.3gcc3.3.2, copy the socket driver files into `$(WIND_BASE)/target/config/wrSbc8641d`.
- b. Copy the socket library header files into `$(WIND_BASE)/target/vThreads/h` (no files should be replaced or overwritten).

## 5.2 Creating Connex Applications

This section contains instructions for creating *Connex* applications for the VxWorks 653 2.3 platforms (sbc8641Vx653-2.3gcc3.3.2 and simpcVx653-2.3gcc3.3.2). The screenshots show the process for sbc8641Vx653-2.3gcc3.3.2.

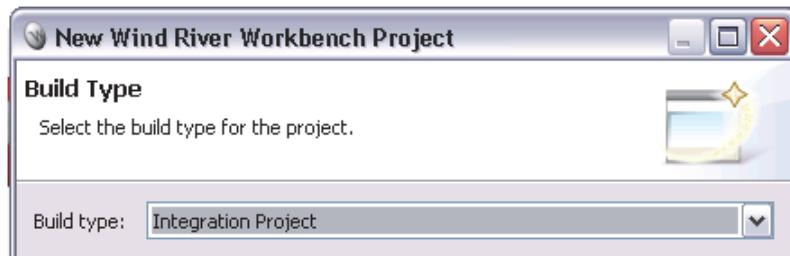
1. Create an integration project with two partitions (one for the publisher, one for the subscriber). Follow the instructions from Wind River for doing this. The following screenshots will guide you through the process.
  - a. Create a new Workbench project.



- b. For the Target operating system, select **VxWorks 653 2.3**.



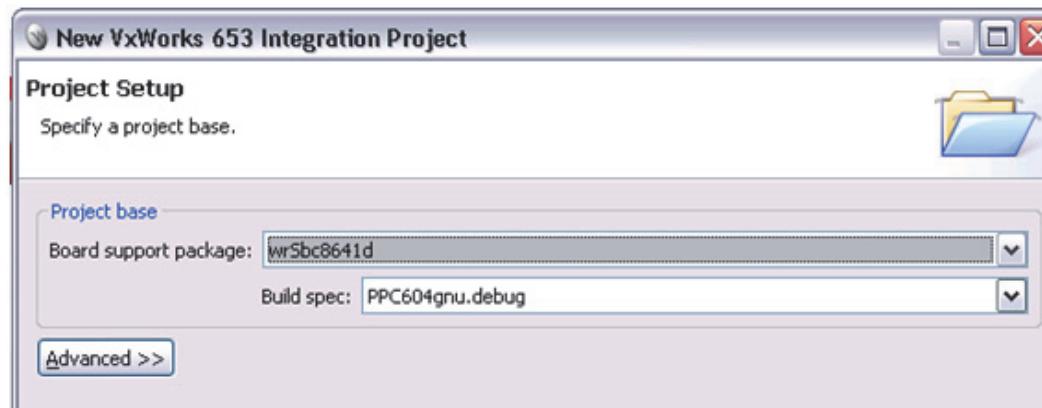
c. For Build type, select **Integration Project**.



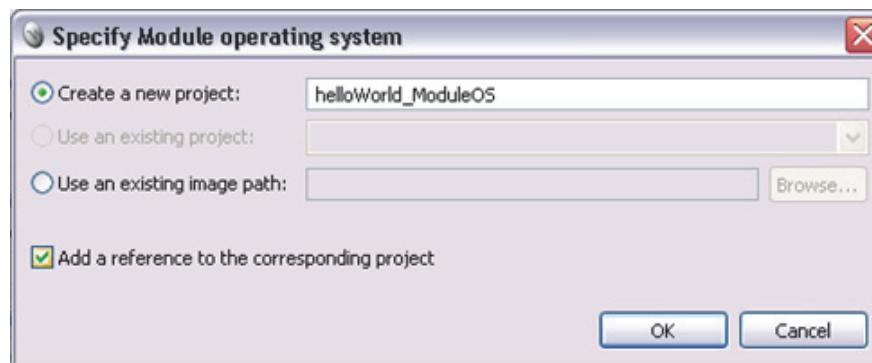
d. Create a project named **helloWorld** in the workspace.



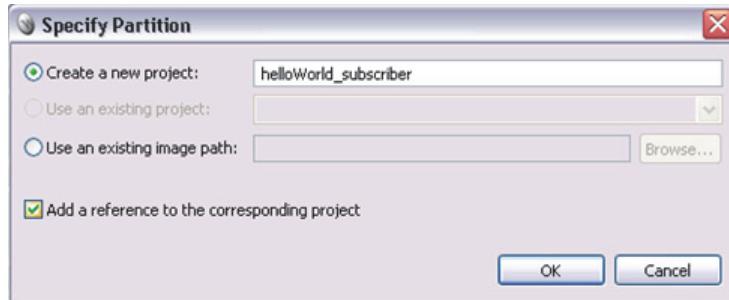
e. Select the appropriate Board Support package. Make sure the debug Build spec is selected. This example assumes the **wrSbc8641d** board support package is selected; alternatively, you could select **simpC**.



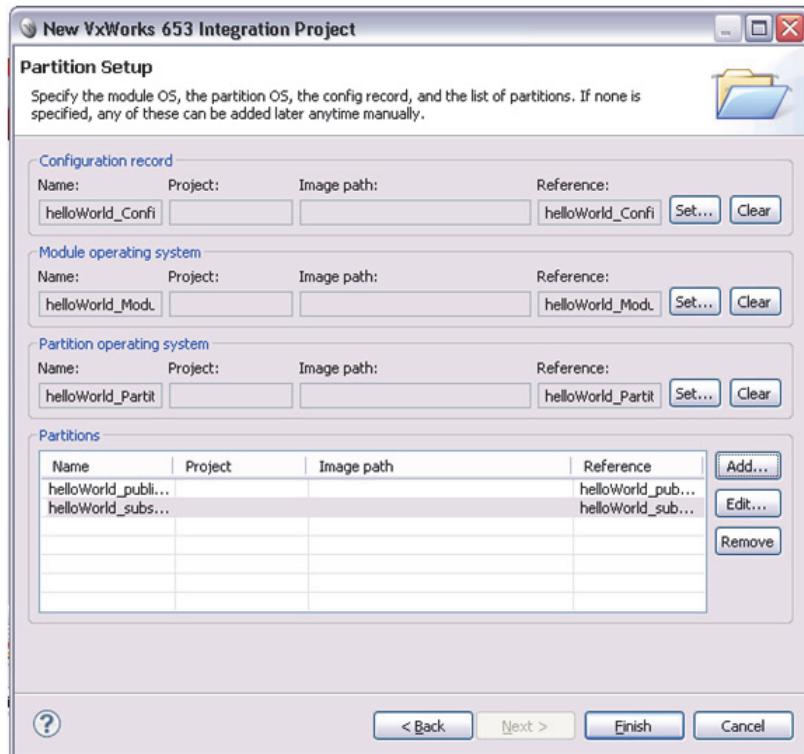
f. Select the default options for adding the ConfigRecord, ModuleOS, and PartitionOS. Make sure the “Add a reference to the corresponding project” checkbox is selected.



g. Create two partitions, **helloWorld\_publisher** and **helloWorld\_subscriber**, to create a Publisher and a Subscriber application, respectively. Make sure the “**Add a reference to the corresponding project**” checkbox is selected.

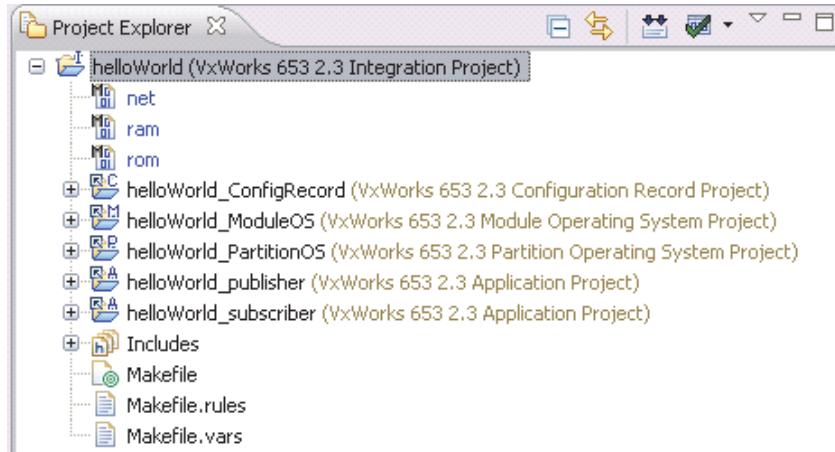


h. Now you are ready to create the Integration Project.



i. Click **Finish** to create the Integration project.

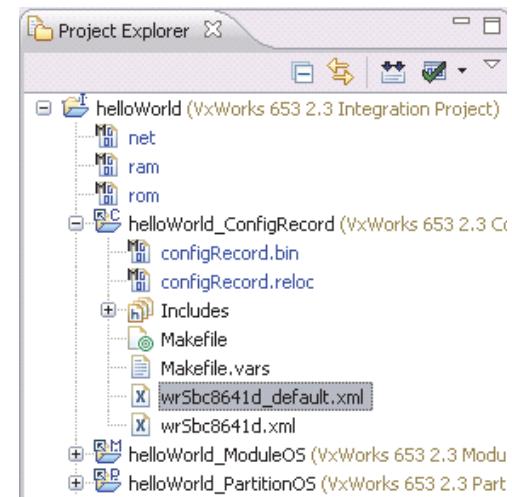
This will create an integration project with **ConfigRecord**, **ModuleOS**, **PartitionOS** and two partitions, **helloWorld\_publisher** and **helloWorld\_subscriber**.



2. Depending on your platform, open either **helloWorld\_ConfigRecord/wrSbc8641d\_default.xml** or **simpc\_default.xml** and make the changes noted below. By default, the file opens in design mode. You may wish to switch to source mode, which makes it easier to copy and paste sections, which is required in later steps.

a. Under Applications:

- i. Change the application name from **wrSbc8641d\_part1** or **simpc\_part1** to **helloWorld\_publisher**.
- ii. In **MemorySize**, make these changes, depending on your platform:



	<b>sbc8641Vx653-2.3gcc3.3.2</b>	<b>simpcVx653-2.3gcc3.3.2</b>
MemorySizeText	0x500000	0x550000
MemorySizeRoData	0x90000	0xe0000
MemorySizeData	0x2000	No change (keep the default value of 0x10000)

For C++ only:

Change the **MemorySize** tag so it ends with '**>**' (not '**/>**')

**For sbc8641Vx653-2.3gcc3.3.2:** Within **MemorySize**, add:

```
<AdditionalSection Name=".gcc_except_table"
Size="0x2000" Type="DATA" />
```

**For simpcVx653-2.3gcc3.3.2:** Within **MemorySize**, add:

```
<AdditionalSection Name=".gcc_except_table"
Size="0x10000" Type="DATA" />
```

Remove **MemorySizePersistentData** and **MemorySizePersistentBss**.

Close **MemorySize** with **</MemorySize>**.

It should look like this when you are done:

**For sbc8641Vx653-2.3gcc3.3.2:**

```
<MemorySize MemorySizeBss="0x1000"
    MemorySizeText="0x500000"
    MemorySizeData="0x2000"
    MemorySizeRoData="0x90000">
    <AdditionalSection Name=".gcc_except_table"
        Size="0x2000" Type="DATA"/>
</MemorySize>
```

**For simpcVx653-2.3gcc3.3.2:**

```
<MemorySize MemorySizeBss="0x10000"
    MemorySizeText="0x550000"
    MemorySizeData="0x10000"
    MemorySizeRoData="0xe0000">
    <AdditionalSection Name=".gcc_except_table"
        Size="0x10000" Type="DATA"/>
</MemorySize>
```

iii. Create a copy of the application **helloWorld\_publisher** and rename it **helloWorld\_subscriber**.

b. Under Partitions:

- i. Change the partition name from **wrSbc8641d\_part1** or **simpc\_part1** to **helloWorld\_publisher**.
- ii. Change the **Application NameRef** from **wrSbc8641d\_part1** or **simpc\_part1** to **helloWorld\_publisher**.
- iii. Under Settings, make these changes, depending on your platform:

	<b>sbc8641Vx653-2.3gcc3.3.2</b>	<b>simpcVx653-2.3gcc3.3.2</b>
RequiredMemorySize	0x1000000	0xF00000
numWorkerTasks	10	10

iv. Create a copy of the partition application **helloWorld\_publisher** and rename it **helloWorld\_subscriber**. Change its ID to 2 and its **Application NameRef** to **helloWorld\_subscriber**.

c. Under Schedules:

- i. Rename **PartitionWindow PartitionNameRef** from **wrSbc8641d\_part1** or **simpc\_part1** to **helloWorld\_publisher**.
- ii. Create a copy of the **PartitionWindow**, and change **PartitionNameRef** to **helloWorld\_subscriber**.
- iii. Add another **PartitionWindow**, with **PartitionNameRef** “**SPARE**” and Duration **0.05**. This partition window schedules the kernel, allowing time in the schedule for system activities like network communications.
- iv. Optionally:
  - (1). If you want only *one* of the applications to run (**helloWorld\_publisher** or **helloWorld\_subscriber**), then you only need a partition window for the one you want to run.

(2). If you do not want the Connexx application to run immediately when the system boots up, change the schedule ID to non-zero and add a SPARE schedule with ID 0.

d. Under HealthMonitor:

- In **PartitionHMTTable Settings**, change **TrustedPartition NameRef** from **wrSbc8641d\_part1** or **simpc\_part1** to **helloWorld\_publisher**. This is an optional field, so it can even be removed from the configuration.
- Optionally, change the **ErrorActions** from **hmDefaultHandler** to **hmDbgDefaultHandler**, in case you want the partitions to stop and not restart on exceptions.

e. Under Payloads:

- Change **PartitionPayload NameRef** from **wrSbc8641d\_part1** or **simpc\_part1** to **helloWorld\_publisher**.
- Create a copy of the **PartitionPayload**, and change **NameRef** to **helloWorld\_subscriber**.

f. Save the changes to **wrSbc8641d\_default.xml** or **simpc\_default.xml**, depending on your platform.

3. For simpcVx653-2.3gcc3.3.2 only:

- Open **helloWorld\_ConfigRecord/simpc.xml**.
- Change the **PhysicalMemory Size** to **0x03000000**.
- In the **ramPayloadRegion** tag, change **Base\_Address** to **0x23000000**.
- Save the changes to **simpc.xml**. After the changes, it should look like this:

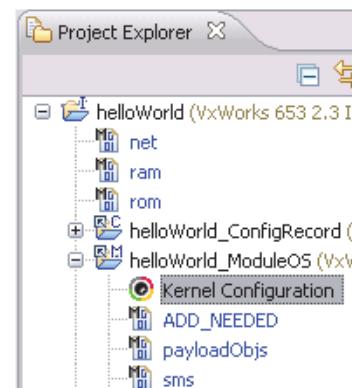
```

<PhysicalMemory Size="0x03000000" Base_Address="0x20000000">
  <kernelMemoryRegion Size="0x00600000" />
  <kernelConfigRecordRegion Size="0x00010000" />
  <kernelPgPool Size="0x00200000" />
  <portRegion Size="0x00200000" />
  <hmLogRegion Size="0x00100000" />
  <ramPayloadRegion Size="0x00000000"
    Base_Address="0x23000000" />
  <aceMemoryRegion Size="0x00000000"
    Base_Address="0x20C00000" />
  <userMemoryRegion Size="0x0b000000"
    Base_Address="0x20C00000" />
</PhysicalMemory>

```

4. Under **helloWorld\_ModuleOS, Kernel Configuration**:

- Include **hardware->peripherals-> BSP configuration variants->socket I/O device [INCLUDE\_SOCKET\_DEV]**.
- Include **development tool components-> debug utilities [INCLUDE\_DEBUG\_UTIL]**. This is needed to enable worker tasks.
- Optionally, include target-resident shell components, and any other components you want to include in the ModuleOS. Note that the target-resident shell component may be too large to include in SimPC without additional memory tuning.

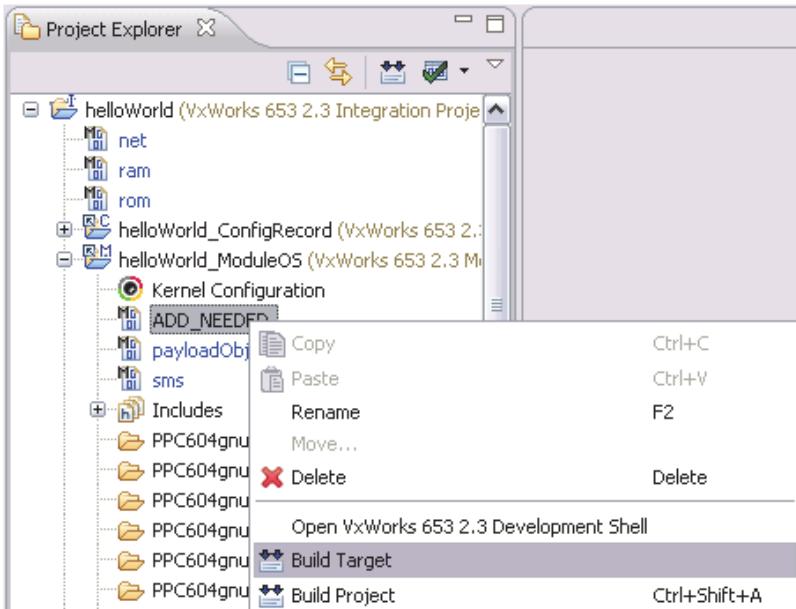




- Save the changes to Kernel Configuration.

**IMPORTANT:** See the *RTI Core Libraries and Utilities Platform Notes (RTI\_CoreLibrariesAndUtilities\_PlatformNotes.pdf)* for a complete list of required kernel components for each platform.

- Build the target **helloWorld\_ModuleOS->ADD\_NEEDED**.



- Generate example code with *rtiddsgen*.

- Create a directory to work in. In this example, we use a directory called **myhello**.
- In the **myhello** directory, create a file called **HelloWorld.idl** that contains a user-defined data type:

```
struct HelloWorld {
    string<128> msg;
};
```

- Use *rtiddsgen* to generate sample code and a makefile, as described in [Generating Code with rtiddsgen \(Section 4.3.2.1\)](#) in the *Getting Started Guide*. Choose either C or C++.

For C:

```
rtiddsgen -language C -example <architecture> HelloWorld.idl
```

For C++:

```
rtiddsgen -language C++ -example <architecture> HelloWorld.idl
```

The supported values for *<architecture>* are listed in the *Release Notes (RTI\_CoreLibrariesAndUtilities\_ReleaseNotes.pdf)*, such as **sbc8641Vx653-2.3gcc3.3.2** or **simpcVx653-2.3gcc3.3.2**.

- Edit the generated example code as described in [Generating Code with rtiddsgen \(Section 4.3.2.1\)](#) in the *Getting Started Guide*.

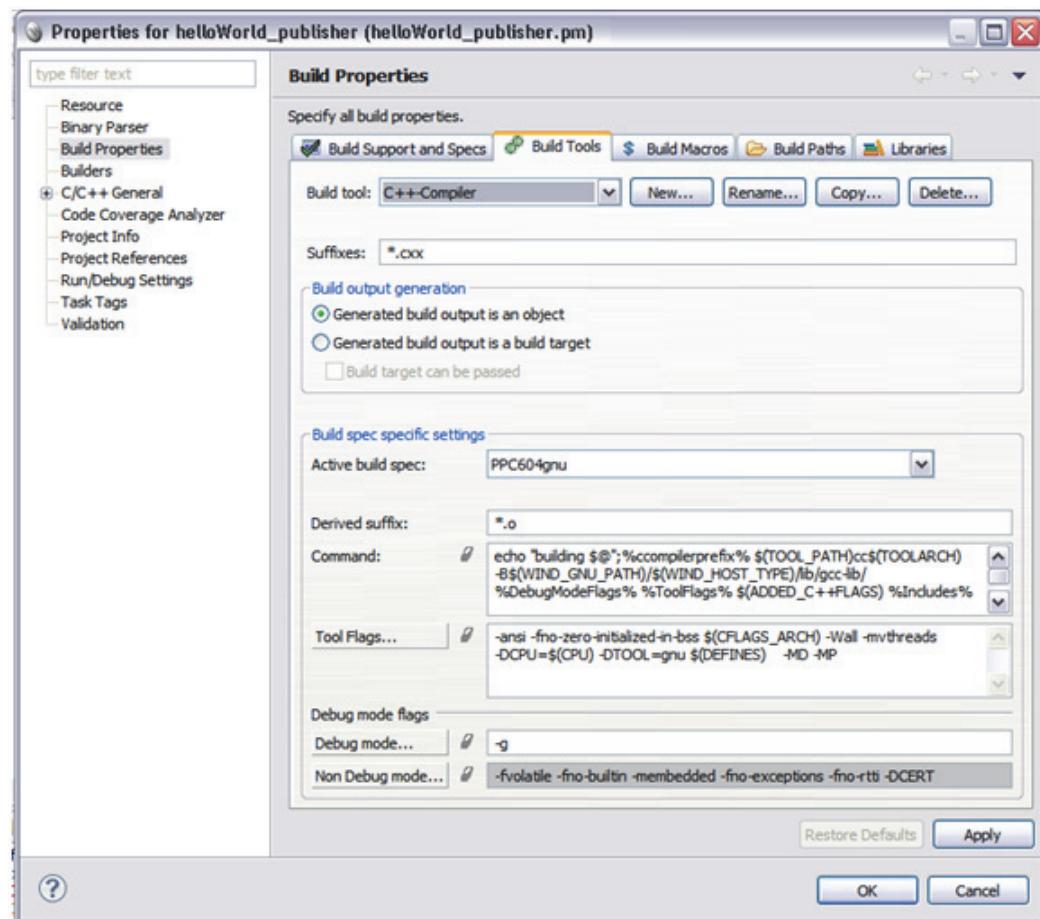
- Import the generated code into the application.

- Right-click **helloWorld\_publisher** and select **Import**.
- In the Import wizard, select **General, File System**, then click **Next**.

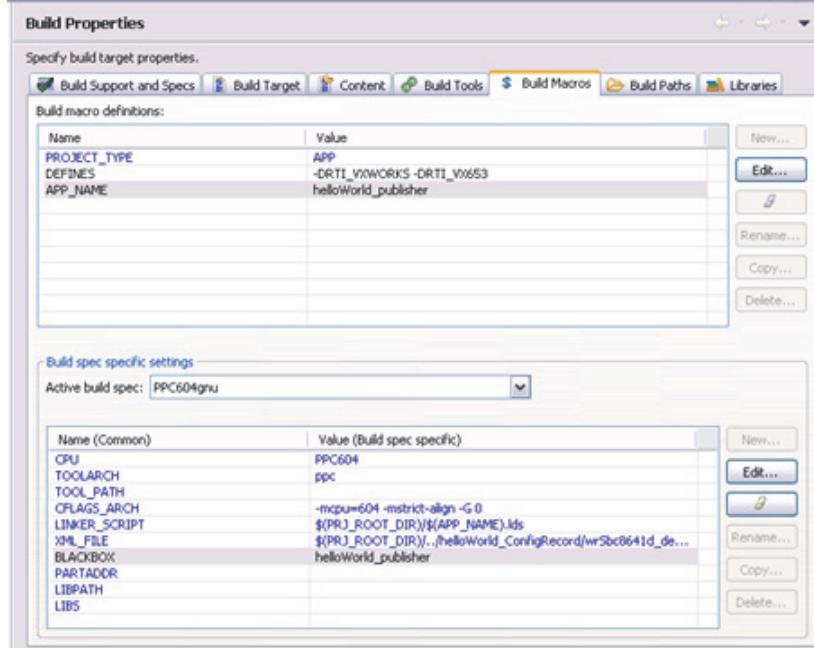
- c. Browse to the **myHello** directory.
- d. Select the generated files, except **HelloWorld\_subscriber**.
- e. Import **sockLib.c** from the socket library into the project.
- f. Right-click **usrAppInit.c** and delete it.
- g. Repeat the same process for **helloWorld\_subscriber**, this time importing **HelloWorld\_subscriber** instead of **HelloWorld\_publisher**.

8. Configure properties for the application.

- a. Right-click **helloWorld\_publisher** and select **Properties**.
- i. Select **Build Properties** in the selection list on the left.
- ii. In the Build Macros tab:
  - Add a new macro, **NDDSHOME**, and set its value to the location where *Connexx* is installed. If this is in a directory with spaces in the path (such as Program Files), put quotation marks around the whole path.
  - Change the **BLACKBOX** value to **helloWorld\_publisher**.
- iii. For C++ only:
  - In the Build Tools tab, select Build tool: **C++-Compiler**.
  - Change Suffixes to **\*.cxx**.
- iv. Click **OK**.



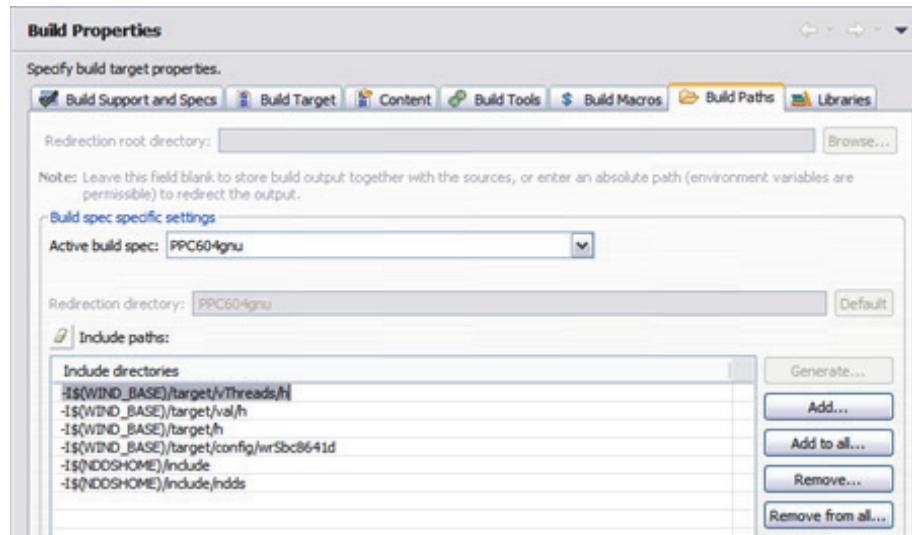
- b. For C: Right-click **helloWorld\_publisher**.  
For C++: Right-click **helloWorld\_publisher**, **Build Targets**, **helloWorld\_publisher-pm**.
- c. Select **Properties**.
- d. In the Build Macros tab, add **-DRTI\_VXWORKS -DRTI\_VX653** to DEFINES.



- e. In the Build Paths tab, select the appropriate 'Active Build Spec' setting (such as PPC604gnu or SIMNTgnu). Then add these include directories, depending on your platform:

sbc8641Vx653-2.3gcc3.3.2	simpCVx653-2.3gcc3.3.2
<code>-I\$(WIND_BASE)/target/config/wrSbc8641d</code>	<code>-I\$(WIND_BASE)/target/config/simpC</code>
<code>-I\$(NDDSHOME)/include</code>	<code>-I\$(NDDSHOME)/include</code>
<code>-I\$(NDDSHOME)/include/ndds</code>	<code>-I\$(NDDSHOME)/include/ndds</code>

For **sbc8641Vx653-2.3gcc3.3.2**, the Build Paths tab will look like this:



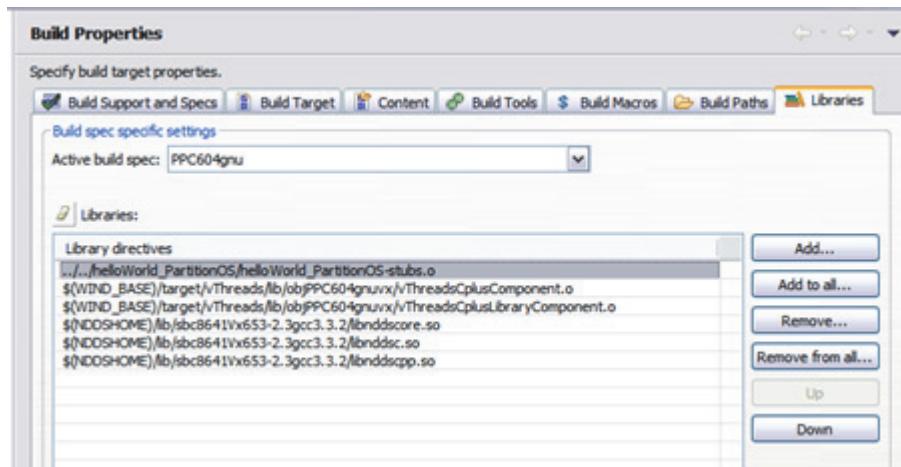
**f. In the Libraries tab:**

Add the following files, depending on your platform and language:

<b>sbc8641Vx653-2.3gcc3.3.2</b>	<b>simpcVx653-2.3gcc3.3.2</b>
For C++ Only: \$(WIND_BASE)/target/vThreads/lib/ objPPC604gnuvx/ vThreadsCplusComponent.o	For C++ Only: \$(WIND_BASE)/target/vThreads/lib/ objSIMNTgnuvx/ vThreadsCplusComponent.o
For C++ Only: \$(WIND_BASE)/target/vThreads/lib/ objPPC604gnuvx/ vThreadsCplusLibraryComponent.o	For C++ Only: \$(WIND_BASE)/target/vThreads/lib/ objSIMNTgnuvx/ vThreadsCplusLibraryComponent.o
For all languages: \$(NDDSHOME)/lib/ sbc8641Vx653-2.3gcc3.3.2/libnddscore.so \$(NDDSHOME)/lib/ sbc8641Vx653-2.3gcc3.3.2/libnddsc.so	For all languages: \$(NDDSHOME)/lib/ simpcVx653-2.3gcc3.3.2/libnddscore.so \$(NDDSHOME)/lib/ simpcVx653-2.3gcc3.3.2/libnddsc.so
For C++ Only: \$(NDDSHOME)/lib/ sbc8641Vx653-2.3gcc3.3.2/libnddscpp.so	For C++ Only: \$(NDDSHOME)/lib/ simpcVx653-2.3gcc3.3.2/libnddscpp.so

**g. Click OK.**

For **sbc8641Vx653-2.3gcc3.3.2**, it should look like this:



h. Repeat the same process for **helloWorld\_subscriber**.

9. Build the Integration Project.

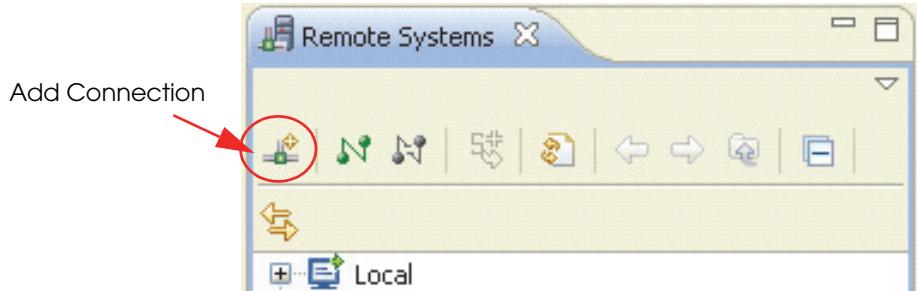
## 5.3 Running Connex Applications on an Sbc8641d Target

1. Boot up your target board with the kernel created by the Integration project.
2. If the *Connex* applications are in schedule 0, they will start up automatically, and you should see the publisher and subscriber communicating with each other.
3. If the *Connex* applications are not in schedule 0, use this command to change to the desired schedule: **arincSchedSet <Schedule number>**.

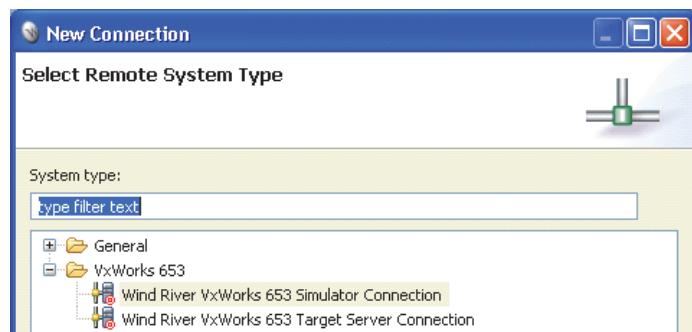
## 5.4 Running Connex Applications on a SimPC Target

1. You can only run the SimPC simulator on Windows XP. If you are running on a newer version of Windows, please install Windows XP mode. Note that this may not work if you have nested virtual machines.
2. You must install the **Ulip** driver to communicate with the SimPC target.
  - a. A network interface allows this application to act as an external machine for network purposes, with its own IP address.
  - b. To install the NDIS driver used by the SimPC target, go to **Control Panel, Network, Adapter, Add, Have Disk**.
  - c. At the prompt for the path to the disk, enter the path to your Wind River VxWorks653 base, followed by **host/x86-win32/bin: \${WIND\_BASE}/host/x86-win32/bin**, then click **OK**.
  - d. Configure the IP address for the driver by going to **Control Panel, Network, Protocols, TCP/IP** and choosing the **Ulip Virtual Driver**.
    - Enter the IP address as **192.168.255.254**, subnet mask **255.255.255.0**, and no default gateway.

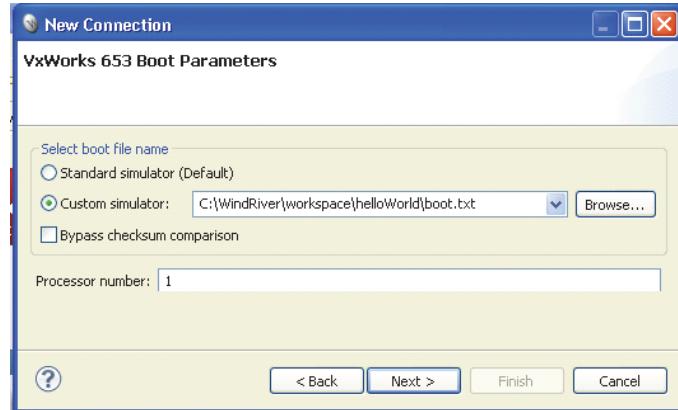
- e. You will need to reboot after installing the NDIS driver.
- 3. Create your connection:
  - a. Click on the add connection button in the Remote Systems panel.



- b. Choose **Wind River VxWorks 653 Simulator Connection** and click **Next**:

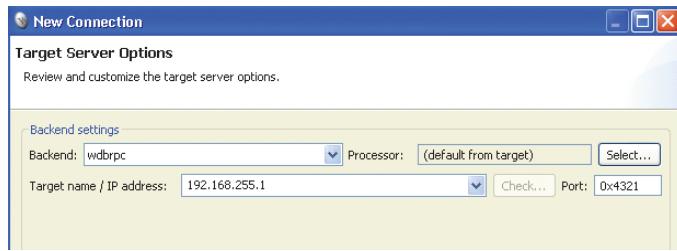


- c. Choose a custom simulator. Point to **boot.txt** in your helloWorld project and click **Next**. If you don't have a **boot.txt** file, this means your build did not complete successfully. Rebuild your Integration project and look for errors.



- d. Click **Next** for the VxWorks Simulator Miscellaneous Options selection, without making any changes.
- e. For the Target Server Options selection, change from **wdbpipe** to **wdbprc**.
  - i. Set the IP address to **192.168.255.simulatorID+1**. For example, if you are creating your first connection, (called **vxsim0\_psc** by default), this IP address will be 192.168.255.1. Simulator IDs range from 0-15.

ii. Set the port number to 0x4321



iii. Continue without making any more changes to the default values, until you click **Finish**.

iv. You can now connect to your VxSim connection by right-clicking and choosing **Connect vxsim0\_psc**.

# Chapter 6 Getting Started on Wind River Linux Systems

This chapter provides instructions on building and running *Connex*t applications on a Wind River Linux system.

This chapter will guide you through the process of compiling and running the Hello World application on a Wind River Linux system.

## In the following steps:

- ❑ Steps 1-5 must be executed on the host machine in a shell that has all the required environment variables. For details, see [Step 1: Set Up the Environment \(Section 3.1.1\)](#) in the [Core Libraries and Utilities Getting Started Guide](#).
- ❑ You need to know the name of your target architecture (look in your %NDDSHOME%\lib directory). Use it in place of <architecture> in the example commands. Your architecture might be 'ppc85xxWRLinux2.6gcc4.3.2'.
- ❑ We assume that you have gmake installed. If you have gmake, you can use the generated makefile to compile. If you do not have gmake, use your normal compilation process. (Note: the generated makefile assumes the correct version of the compiler is already in your path and that NDDSHOME is set.)

## To create the example applications:

1. Create a directory to work in. In this example, we use a directory called **myhello**.
2. In the myhello directory, create a file called **HelloWorld.idl** that contains a user-defined data type:

```
struct HelloWorld {  
    string<128> msg;  
};
```
3. Use **rtiddsgen** to generate sample code and a makefile as described in [Generating Code with rtiddsgen \(Section 4.3.2.1\)](#) in the [Core Libraries and Utilities Getting Started Guide](#). Choose either C or C++.

For C:

```
rtiddsgen -language C -example <architecture> HelloWorld.idl
```

For C++:

```
rtiddsgen -language C++ -example <architecture> HelloWorld.idl
```

Edit the generated example code as described in [Generating Code with rtiddsgen \(Section 4.3.2.1\)](#) in the [Core Libraries and Utilities Getting Started Guide](#).

---

4. Set up your environment with the **wrenv.sh** script in the Wind River Linux base directory.

```
wrenv.sh -p wrlinux-3.0
```

5. With the **NDDSHOME** environment variable set, build the Publisher and Subscriber modules using the generated makefile.

```
make -f makefile_HelloWorld_<architecture>
```

After compiling, you will find the application executables in **myhello/objs/<architecture>**.

6. Connect to the Wind River Linux target (using telnet, ssh, serial console, connection manager, etc.) and start the subscriber application, **HelloWorld\_subscriber**.

```
HelloWorld_subscriber
```

In this shell, you should see that the subscriber is waking up every 4 seconds to print a message:

```
HelloWorld subscriber sleeping for 4 sec...
HelloWorld subscriber sleeping for 4 sec...
HelloWorld subscriber sleeping for 4 sec...
```

7. Connect to the Wind River Linux target and start the publisher application, **HelloWorld\_publisher**.

```
HelloWorld_publisher
```

In this second (publishing) shell, you should see:

```
Writing HelloWorld, count 0
Writing HelloWorld, count 1
Writing HelloWorld, count 2
```

8. Look back in the first (subscribing) shell. You should see that the subscriber is now receiving messages from the publisher:

```
HelloWorld subscriber sleeping for 4 sec...
msg: "Hello World! {0}"
HelloWorld subscriber sleeping for 4 sec...
msg: "Hello World! {1}"
HelloWorld subscriber sleeping for 4 sec...
```

## Chapter 7 Getting Started on Wind River VxWorks MILS 2.1.1 Systems



**Important Note:** To use *RTI Connex* on a VxWorks MILS 2.1.1 system, you must have the patch provided by Wind River that corrects defect number WIND00343321. You can obtain this patch through the regular Wind River support channel.

This chapter provides instructions to configure a complete MILS 2.1.1 system image with an application that uses *Connex*. Please refer to the documentation provided by Wind River for more information on the MILS system; you should also refer to the *Core Libraries and Utilities Platform Notes* ([RTI\\_Connex\\_PlatformNotes.pdf](#)).

This chapter will guide you through the process of generating, compiling, and running a “Hello, World” application on VxWorks MILS 2.1.1 systems by expanding on [Section 4.3.2.1 in the Getting Started Guide](#); please read the following alongside that section.

The instructions in this chapter use Wind River Workbench to create the MILS system image. The overview of the workflow includes:

- [Step 1: Generate Connex Support Files and Example with rtiddsgen \(Section 7.1\)](#)
- [Step 2: Create a VxWorks GuestOS Application Project \(Section 7.2\)](#)
- [Step 3: Create a VxWorks MILS Integration Project \(Section 7.3\)](#)
- [Step 4: Integrate GuestOS Application Project and Generated rtiddsgen Files into MILS Integration Project \(Section 7.4\)](#)
- [Step 5: Deploy MILS Image to Target \(Section 7.5\)](#)

## 7.1 Step 1: Generate Connexx Support Files and Example with `rtiddsgen`

1. Given a sample file with an IDL definition, obtain the *Connexx* support files and example by following the steps in [Section 4.3.2.1 in the \*Getting Started Guide\*](#).

After you have completed this step, you will end up with source files and headers that implement the type support for your IDL definition, as well as an example publisher and an example subscriber for the type.

Eventually, when we create our MILS application, we will call the `publisher_main` or `subscriber_main` functions from it, so make sure the `publisher_main` or `subscriber_main` functions are not declared as "static" (modify the example publisher and subscriber if you need to by simply removing the "static" qualifier from their function definitions if they have it).

2. If you are using C++, rename all `.cxx` files to `.cpp`.

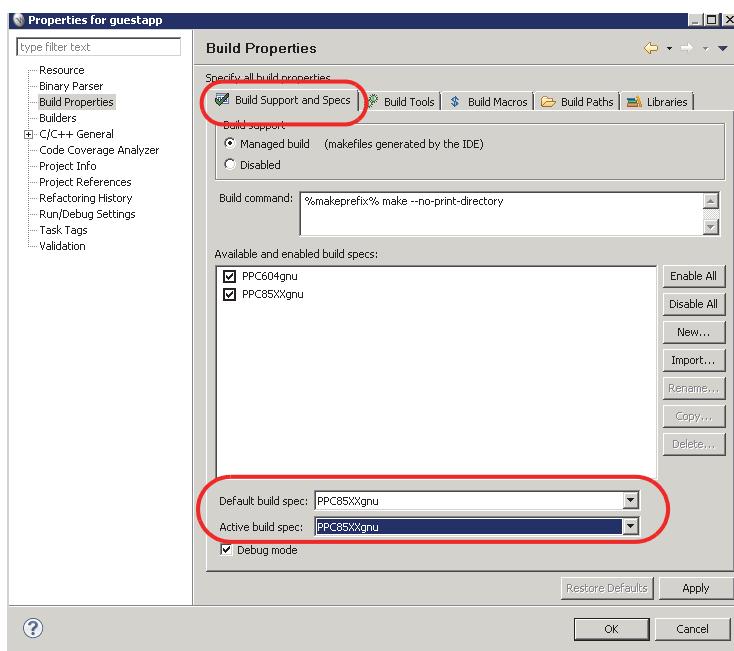
---

## 7.2 Step 2: Create a VxWorks GuestOS Application Project

1. From the **File** menu, select **New, Wind River Workbench Project**.
2. In the resulting dialog, select **VxWorks MILS VxWorks Guest OS 2.2.3.1** and click **Next**.



3. In the **Build Type** dialog, select **Application** and click **Next**.
4. For the project name, type **guestapp**, and click **Finish**.
5. Right-click on the project and select **Properties**.
6. In the left pane of the **Properties** dialog, select **Build Properties**.
7. In the **Build Support and Specs** tab, set the **Default build spec** and the **Active build spec** to **PPC85XXgnu**.



8. In the Build Macros tab:
  - a. Under **Build Macro Definitions**, set **DEFINES** to **-DRTI\_VXWORKS**.
  - b. Define a new build macro named **NDDSHOME** (click **New** to add it). Set its value to the path where *RTI Connexx* is installed. If you are using a Windows system, use quotes around the path (for example, the value could be **"C:\Program Files\RTI\ndds.4.5f"**).
  - c. Set **Active build spec** to **PPC85XXgnu**.
  - d. In **CFLAGS\_ARC**, add **-mlongcall** to the end (leave the rest of the flags in that cell as is).
  - e. Set **LIBPATH** to **-L\$(NDDSHOME)/lib/<architecture>**.

f. If you will *not* be using C++: set **LIBS** to: **-lnddscz -lnddscorez**

If you *will* be using C++: set **LIBS** to: **-lnddscppz -lnddscz -lnddscorez**

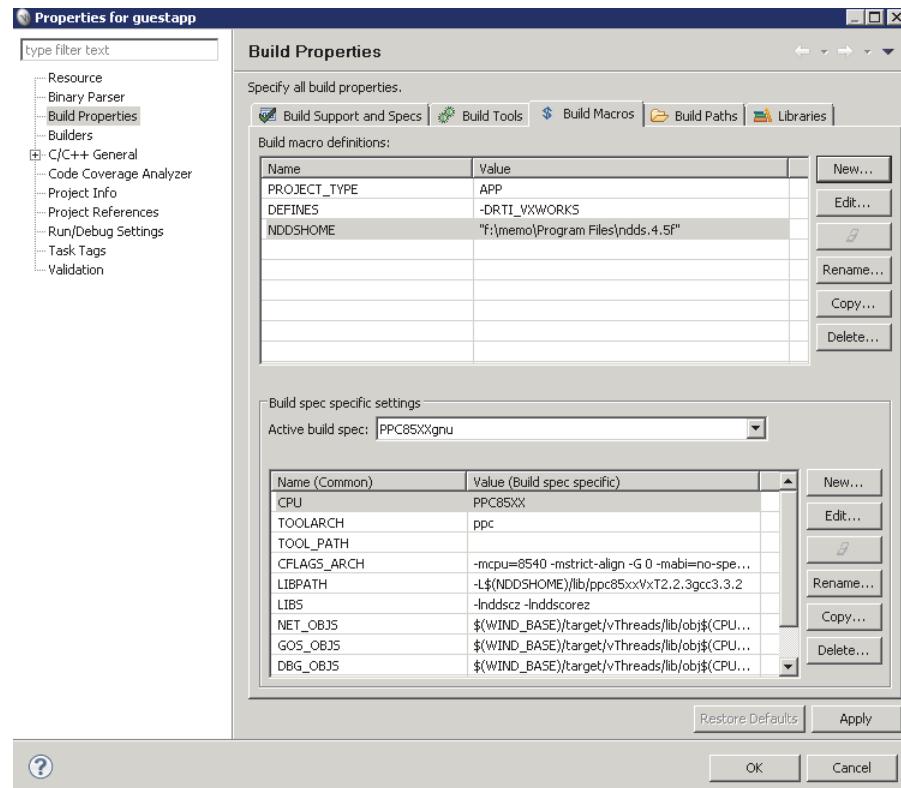
g. Set **NET\_OBJS** to the following (all in one line):

```
$ (WIND_BASE)/target/vThreads/lib/obj$(CPU)gnuvx/vThreadsNetwrsComponent.o
$ (WIND_BASE)/target/vThreads/lib/obj$(CPU)gnuvx/vThreadsNetinetComponent.o
$ (WIND_BASE)/target/vThreads/lib/obj$(CPU)gnuvx/vThreadsNetCommonComponent.o
$ (WIND_BASE)/target/vThreads/lib/obj$(CPU)gnuvx/vThreadsNetBufCommonComponent.o
$ (WIND_BASE)/target/vThreads/lib/obj$(CPU)gnuvx/vThreadsNetUtilComponent.o
$ (WIND_BASE)/target/vThreads/lib/obj$(CPU)gnuvx/avlLib.o
```

h. If you will *not* be using C++: no changes are needed to **GOS\_OBJS**.

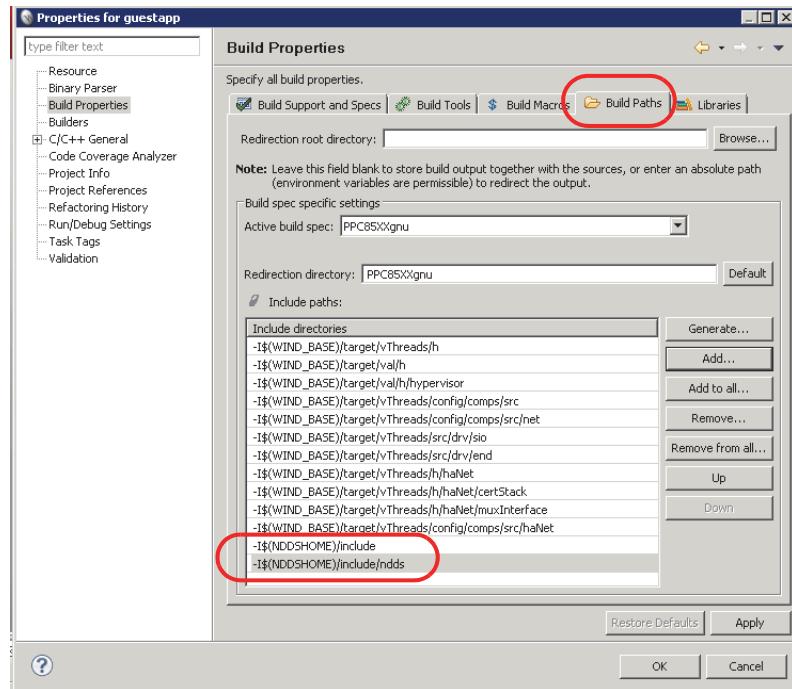
If you *will* be using C++: append the following to **GOS\_OBJS** (all in one line):

```
$ (WIND_BASE)/target/vThreads/lib/obj$(CPU)gnuvx/vThreadsCplusComponent.o
$ (WIND_BASE)/target/vThreads/lib/obj$(CPU)gnuvx/vThreadsCplusLibraryComponent.o
```



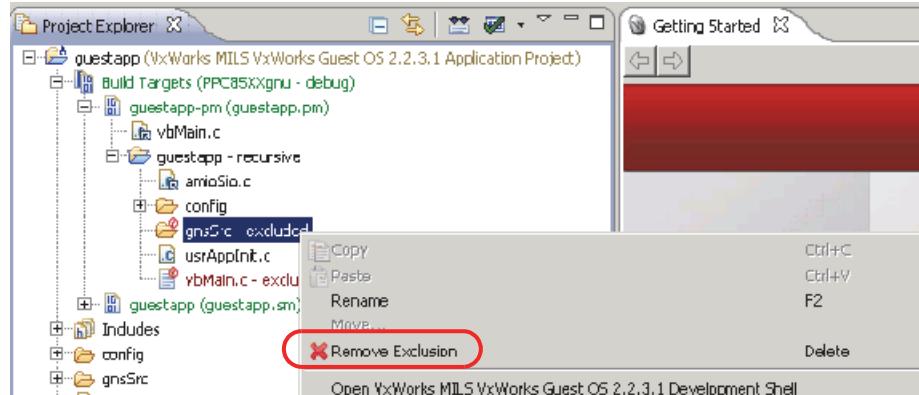
9. In the Build Paths tab:

- Add `-I$(NDDSHOME)/include`
- Add `-I$(NDDSHOME)/include/ndds`.

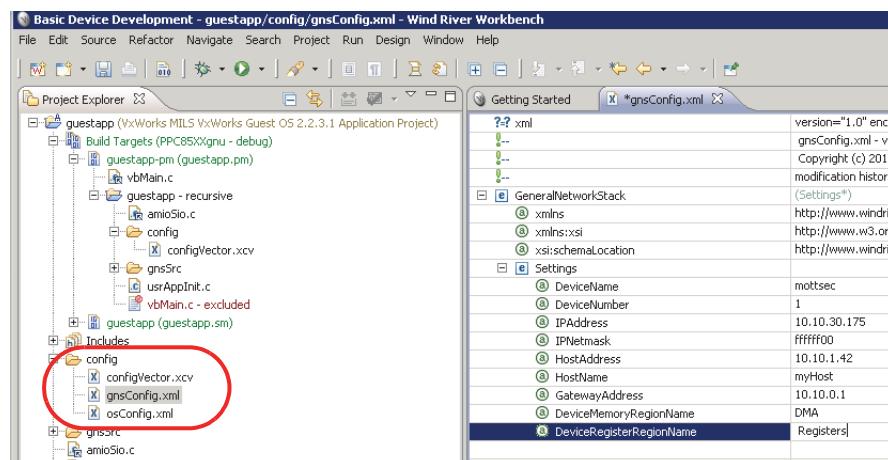


- Click **OK**. If you see a prompt about rebuilding the C/C++ index, click **Yes**.

10. In the Project Explorer pane on the left, expand **Build Targets**, expand **<project\_name>-pm**, and finally expand **<project\_name>-recursive**. Then right-click on the **gnsSrc-excluded** item and remove the exclusion for it. Leave the rest in the default include/exclude mode.



11. Configure the network for your setup (open the top-level **config** directory of your VxWorks GuestOS project and edit the **gnsConfig.xml** file to match your network setup).



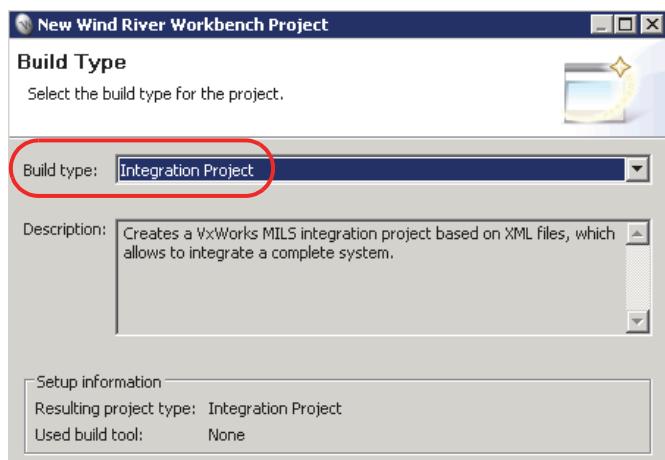
12. Right-click on the project and build it. If you see a dialog asking if you want to set the include search path, click **Continue**.

## 7.3 Step 3: Create a VxWorks MILS Integration Project

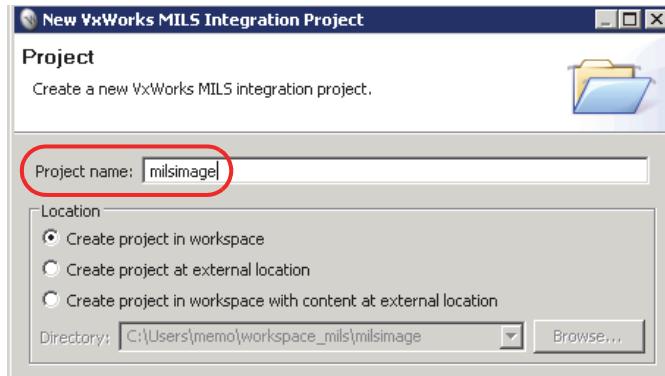
1. From the **File** menu, select **New, Wind River Workbench project**.
2. In the resulting dialog, select **VxWorks MILS 2.1.1**.



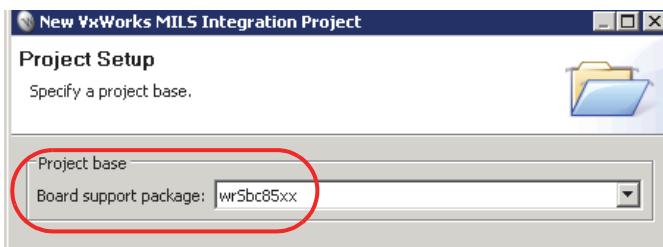
3. For **Build Type**, select **Integration Project** and click **Next**.



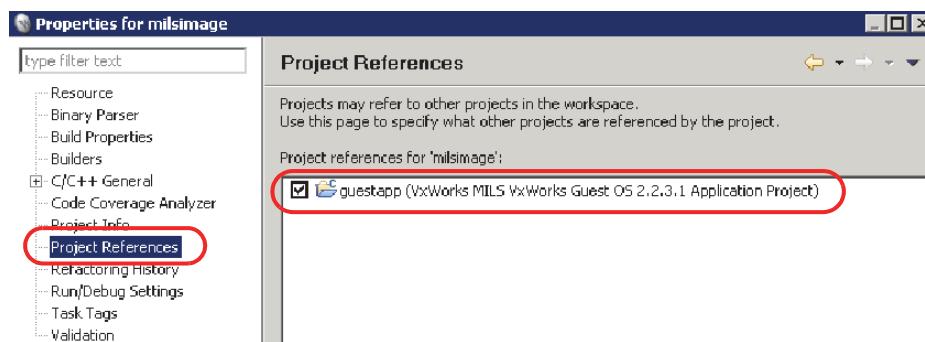
4. In the Project dialog: for **Project name**, type **milsimage** and click **Next**.



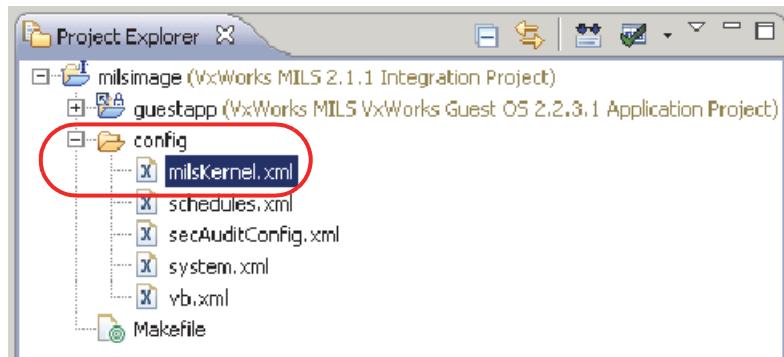
5. In the Project Setup dialog: for **Board Support Package**, select **wrSbc85xx** and click **Finish**.



6. Right-click on the newly created **milsimage** project and select **Properties**.
7. In the Project References section, add **guestapp** (the VxWorks GuestOS project that you created earlier). Once you accept these changes, you should see your **milsimage** and **guestapp** projects merge into a single entity in the Project Explorer.



8. In the Project Explorer pane on the left, navigate to the **config** directory of the **milsimage** project, where you will find the file, **milsKernel.xml**.

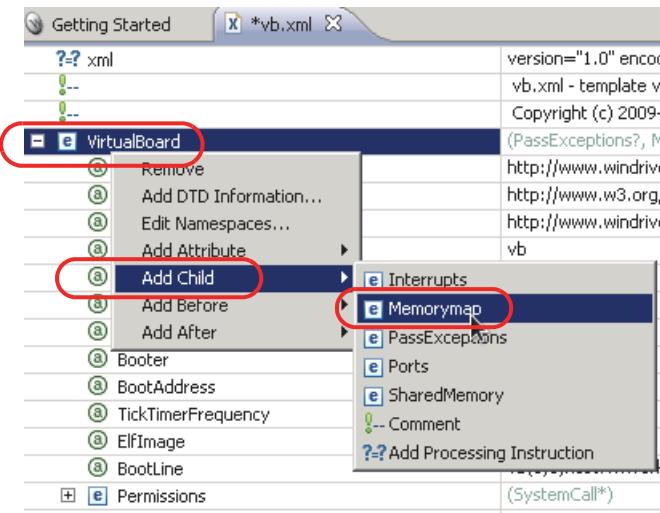


9. Double-click on **milsKernel.xml** and use the XML editor to make these changes:
  - Under MILS Kernel, set **RamSize** to **0xA00000**.
  - Under RamPayload, set **RamPayloadSize** to **0x17000000**.
  - Under PcbMemPool, set **PcbPoolAddr** to **0xA00000**.
  - Under PayloadsMemPool, set **PayloadsMemPoolAddr** to **0xB00000**.
  - Under PayloadsMemPool, set **PayloadsMemPoolSize** to **0x8000000**.
  - Under SharedMemPool, set **SharedMemPoolAddr** to **0x8B00000**.
  - Under SharedMemPool, set **SharedMemPoolSize** to **0x4000**.

**Note:** After changing the value of a cell, you must move to another cell so that your change will be picked up.

10. In the Project Explorer pane on the left, navigate to the **config** directory of the **milsimage** Integration project, where you will find the file, **vb.xml**.
11. Double-click on **vb.xml** and use the XML editor to make these changes:
  - Under VirtualBoard, set **RamSize** to **0x8000000**.
  - Under VirtualBoard, set **ElfImage** to **guestapp.sm**. Note that the file extension is **.sm**, not **.pm**. Also, if you used a different name for the GuestOS application project, you will need to modify this value accordingly.

- Right-click on the **VirtualBoard** element and select **Add Child, Memory Map** (because we need to map in some devices).



**12.** In the newly created Memory-Map element:

- Set **NumMemoryRegions** to 3.
- Right-click on the **Memory Map** element and select **Add Child, Region**; do this three times to add three regions.

Make the following changes in each region:

	Region 1	Region 2	Region 3
<b>Name</b>	Uart	Tsec	nvRam
<b>MmuCacheAttr</b>	0xF36	0xF36	0xF36
<b>VirtualAddress</b>	0xD0000000	0xE0024000	0xF8B00000
<b>Length</b>	0x1000	0x1000	0x01000
<b>PhysicalAddress</b>	0xE0004000	0xE0024000	0xF8B00000

13. Save your changes (be sure to switch to another cell after you edit each cell's contents and before closing the file, so it registers all your changes).

vb.xml	
?? xml	version="1.0" encoding="UTF-8"
!--	vb.xml - template virtual board configuration
!--	Copyright (c) 2009-2010 Wind River Systems
VirtualBoard	(PassExceptions?, Memorymap?, Interrupts?,
@ xmlns	http://www.windriver.com/vxWorksMILS
@ xmlns:xsi	http://www.w3.org/2001/XMLSchema-instance
@ xsi:schemaLocation	http://www.windriver.com/vxWorksMILS
@ Name	vb
@ Id	1
@ RamSize	0x8000000
@ BoardConfig	0xFF000000
@ Booter	vbBootElf.bin
@ BootAddress	0xF0000000
@ TickTimerFrequency	10
@ ElfImage	guestapp.sm
@ BootLine	vb(0,0)host:vxWorks h=90.0.0.3 e=90.1
Memorymap	(Region*)
@ NumMemoryRegions	3
Region	
@ Name	Uart
@ MmuCacheAttr	0xF36
@ VirtualAddress	0xD0000000
@ Length	0x1000
@ PhysicalAddress	0xE0004000
Region	
@ Name	Tsec
@ MmuCacheAttr	0xF36
@ VirtualAddress	0xE0024000
@ Length	0x1000
@ PhysicalAddress	0xE0024000
Region	
@ Name	nvRam
@ MmuCacheAttr	0xF36
@ VirtualAddress	0xF8B00000
@ Length	0x01000
@ PhysicalAddress	0xF8B00000
Permissions	(SystemCall*)

14. Open the **integration** project's **Makefile** (it should be under the top-level **milsimage** project; do not confuse it with the **guestapp** project's **Makefile**).

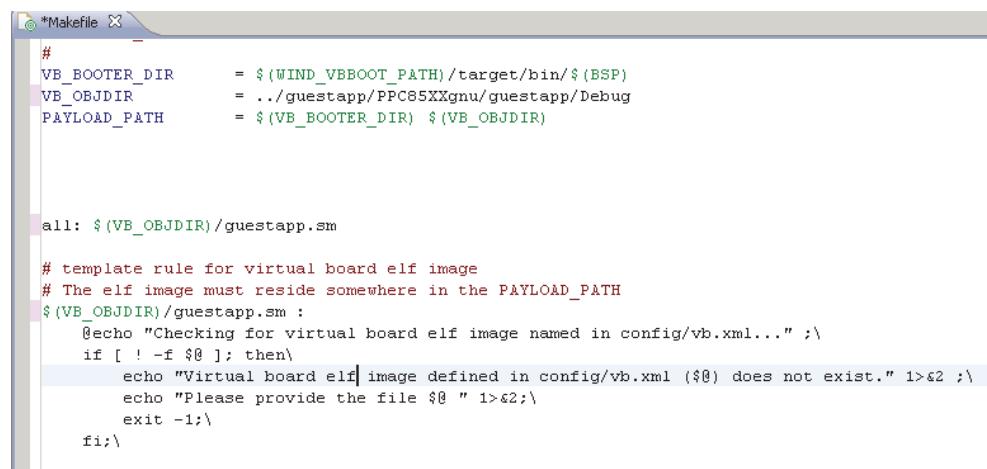
15. Update the **VB\_OBJDIR** make variable to:

```
./guestapp/PPC85XXgnu/guestapp/Debug
```

This way we point to the output of the **guestapp** application project.

16. Update the **all** make target to **\$(VB\_OBJDIR)/guestapp.sm**.

17. Update the rule after the **all** target so it also references **guestapp.sm** instead of **hello.sm**.



```
*Makefile x
#
VB_BOOTER_DIR      = $(WIND_VBBOOT_PATH)/target/bin/$(BSP)
VB_OBJDIR          = ./guestapp/PPC85XXgnu/guestapp/Debug
PAYLOAD_PATH        = $(VB_BOOTER_DIR) $(VB_OBJDIR)

all: $(VB_OBJDIR)/guestapp.sm

# template rule for virtual board elf image
# The elf image must reside somewhere in the PAYLOAD_PATH
$(VB_OBJDIR)/guestapp.sm :
    @echo "Checking for virtual board elf image named in config/vb.xml..." ;\
    if [ ! -f $0 ]; then \
        echo "Virtual board elf image defined in config/vb.xml ($0) does not exist." 1>&2 ;\
        echo "Please provide the file $0" 1>&2 ;\
        exit -1; \
    fi;
```

---

## 7.4

## Step 4: Integrate GuestOS Application Project and Generated rtiddsgen Files into MILS Integration Project

1. Import the source files that you generated from your IDL file into the project:
  - a. Right-click on the **guestapp** project and select **General, File System**. Navigate to the directory that contains your generated files.
  - b. Click on the directory's name in the left pane of the resulting dialog box and check all the C/C++ source files and header files from that directory.
  - c. Click **Finish**.

2. If you are using C++, rename all imported .cxx files to .cpp if you haven't already done so.
3. Make sure you have removed the **static** qualifier from the signature of the functions **publisher\_main** and **subscriber\_main** if they had it. These functions would be in the imported **<idl\_struct\_name>publisher.c** and **<idl\_struct\_name>subscriber.c** files, respectively. The objective is to make them callable from outside these files.
4. Using the Project Explorer in the left pane of WorkBench, navigate to the file **usrAppInit.c** in the **guestapp** project. Double-click to edit the file and replace its entire contents with the following:

```
#include <stdio.h>
#include <taskVarLib.h>
#include <muxLib.h>
#include <bootLib.h>
#include <routeLib.h>
#include <netShow.h>
#include <usrLib.h>

/* defines */
#undef DEBUG

/* globals */
UINT32 boardNum = 0;
extern BOOT_PARAMS sysBootParams;
extern void usrNetworkInit (void);
extern int publisher_main(int domainId, int sample_count);

void usrAppInit (void)
{
    char dev[END_NAME_MAX + 2]; /* device name + unit */

    taskVarInit( );
    boardNum = vbConfig->boardID;

    /* avoid startup messages in console */
    taskDelay (sysClkRateGet () * 10);

    printf ("\n\n*** MILS User Space RTI App.***\n\n");
    printf ("On Virtual Board %d\n\n",boardNum);

    /* start the network */
    usrNetworkInit ();
}
```

```
routeAdd ("0.0.0.0", sysBootParams.ead);

taskDelay (sysClkRateGet () * 1);

printf ("\n\n    RTI App Starting    \n\n");
sprintf (dev, "%s%d\n", sysBootParams.bootDev,
        sysBootParams.unitNum);
printf ("before ifShow\n");
ifShow (dev);
printf ("after ifShow\n");

muxShow (sysBootParams.bootDev, sysBootParams.unitNum);

printf ("\nAPPLICATION: Launching\n\n");
taskSpawn ("pub", 75, 0, 0x20000, (FUNCPTR) publisher_main,
           0, 100, 0, 0, 0, 0, 0, 0, 0, 0, 0);
while (1){
    taskDelay (sysClkRateGet () * 60);
    printf ("\nVirtual Board %d is alive.\n", boardNum);
    /*memShow(0); */
}
}
```

5. Build the MILS Integration project: right-click the **milsimage** project and select **Rebuild** from the context menu. The build should complete with no errors.

---

## 7.5 Step 5: Deploy MILS Image to Target

Once you have completed Step 4, you should have a MILS image file in your file system. The location where you can find the image file relative to the MILS integration project is: **obj\_wrSbc85xx/milsKernel.elf**.

Upload this **.elf** image file to your target. One way to do this is to upload the file to a tftpserver that is accessible from your target board, then have the target board pull the image over the network. Boards with a VxWorks boot loader can do this in a standard way; consult the board's documentation for further information.

Once you have deployed the MILS image to your board, it will start publishing samples with your IDL definition as a data type. It will print out to the target's console as it publishes samples. You can start a subscriber in the same domain on other computers connected to the same network to verify the samples are being sent.